UNIVERSITY OF TWENTE.

Master of Science Thesis

Real-World Analyses of Internet Protocols

Author:
Jurgen Gijs van den Broek

Committee:
dr.ir. Aiko Pras
ir. Roland van Rijswijk
dr. Anna Sperotto

Faculty: Electrical Engineering, Mathematics and Computer Science
Group: Design and Analysis of Communication Systems

July 27th, 2012

"So, what's it like in the real world? Well, the food is better, but beyond that, I don't recommend it."

Bill Watterson (American author)

UNIVERSITY OF TWENTE

Abstract

Master of Science

Real-World Analyses of Internet Protocols

by Jurgen Gijs VAN DEN BROEK

This thesis presents research work as part of the author's Master of Science final research assignment involving the analyses of real-world internet protocol. This thesis encompasses five publications, including papers, an IETF Internet-Draft and a poster.

The first two publications are a pending journal publication and a poster presenting findings following research in the field of the Domain Name System (DNS) at SURFnet. This includes observations of real-world DNS usage and presents two solutions to improve response deliverability in DNSSEC. These solutions have also been tested in the production environment of SURFnet.

The remaining three publications result from the research following the author's Bachelor of Science degree. This involves definitions for traffic in real-world Simple Network Management Protocol (SNMP) network traces. The final paper discusses separating periodic from aperiodic SNMP traffic, based on the definitions stated in the first two publications.

Acknowledgements

This work concludes a phase in my life as a Master student at the University of Twente. I would like to thank a number of people who have supported me during this phase.

Firstly, I wish to express my sincere gratitude to my supervisors Aiko Pras, Roland van Rijswijk and Anna Sperotto. Aiko has also been my supervisor for my final Bachelor of Science assignment. I want to thank Aiko for his constructive input and guidance during the course of my research period, but also for our pleasant meetings in the past years. Our meetings easily lasted hours, during which we extensively discussed political developments in the world, besides my assignment. His support during my period of sickness is also deeply appreciated. I would like to express my gratitude to Roland for his constructive and critical views, for making time for our regular meetings and for giving me the opportunity to attend multiple conferences and activities what were not necessarily limited to the scope of my assignment. I thank Anna for her invaluable guidance during some research challenges. Her expertise and assistance allowed me to take the right decisions during the research phase.

Secondly, I would like to thank the co-authors and reviewers of the publications. Their comments and suggestions have allowed me to improve the publications of my work.

Finally, I would like to especially thank my parents (Eltjo and Ria), my brothers (Philip and Rutger) and my friends for their interest and support during the years of my study and my sickness. I would like to thank Jan Willem also for designing the cover of this thesis.

Jurgen Gijs van den Broek
Utrecht
July 27th, 2012

Contents

Chapter 1

Introduction

This thesis covers the research results as part of the author's Master of Science graduation project. The research results are covered by a total of four papers and a poster, of which two papers are pending publication. This research focuses on the real-world analysis of two internet protocols: the Simple Network Management Protocol (SNMP) and the Domain Name System (DNS).

The first part of this thesis describes the results related to a research assignment at SURFnet, the Dutch National Research and Education Network provider. This assignment is in the field of DNS, which started in September 2011 and finished in July 2012. Preliminary results of this work have been presented at a TERENA Networking Conference (TNC) in the form of a poster. A paper containing the final results is pending publication.

The second part involves SNMP research. This is a continuation of the author's final research assignment for his Bachelor of Science degree. This resulted in two publications, including an IETF Internet-Draft and an article in the proceedings of the second AIMS conference. A third paper is pending publication.

All work related to this thesis has been performed between early 2008 and July 2012. This period is longer than average for a Master of Science graduation project, due to the nature of the assignment, but also due to personal circumstances of the author between 2009 and 2011.

The structure of this thesis is not common, because of the special nature of this graduation project. This thesis describes the graduation project assignment, supported by the (pending) publications. The conclusions of this graduation project are the individual conclusions of the publications.

The remainder of this chapter describes the graduation project, the research topics and gives an overview of the organisation of this thesis.

1.1 Assignment

The main goal of a Master of Science graduation project is to allow the student to demonstrate the abilities required for a Master of Science degree. The criteria for graduation that must be met are stipulated in the Studiegids Master CS 2011-2012[1] of the EWI faculty at the University of Twente. This section discusses those criteria and how they are met by the author and this thesis.

A traditional Master of Science graduation project consists of a clearly defined set of research questions, literature research, a phase leading to the actual research contribution and the formalisation of the results in a thesis. The research assignment of the author consists of independent research in two different subjects within the Telematics domain, the writing of five publications in the form of an Internet-Draft, three papers, a poster and presenting results at multiple meetings and conferences. This thesis combines these publications. At the moment of writing are three of the five publications already published and have been presented at conferences where applicable. Two papers are pending publication.

The following discusses each of the criteria that must be met in accordance with the Studiegids Master CS 2011-2012. It also states how this thesis fulfils each of these criteria.

- **Clearly formulate a problem statement**
  Defining a problem clearly and stating a set of research questions to solve the problem needs to be achieved by the student. The DNS related research at SURFnet encompasses the first part of this thesis. The first part of this research phase involved the definition of the problem statement and a set of research questions. The results of this initial phase have been presented as part of a preparation course before the actual research in this subject started.

  The work in the SNMP subject follows from the author's final research assignment for his Bachelor of Science degree, where a clear problem statement was already required. Still, the publications involved are a continuation of that work and required a clear problem statement with their own research questions. The problem statements defined in those publications are different, though they follow from previous research.

- **Identify relevant literature**
  Each publication requires a clear understanding of related work in the field. As a preparation for each publication, but also before each research phase per subject, an extensive amount of literature was researched. Not only allows this the researcher to develop an understanding of the research field,

[1] `http://www.utwente.nl/ewi/onderwijs/studiegidsen/master/studiegids_master_cs.pdf`

but also to identify the related work. Identifying relevant literature was necessary for each publication, but also more generally at the start of both research subject: SNMP and DNS.

- **Draw up a work plan**
  Researching and documenting results needs to be performed based on properly defined plans that indicate when certain milestones need to be met and when the final results are due.

  The DNS research started with preparation work resulting in a problem statement and results from literature research. This phase also yielded a work plan containing a clear set of research questions that were to be answered in a specific order. This work plan also contained a final date at which the final results were expected. Also, there were submission deadlines for the individual publications that had to be adhered to. In addition to that, there were regular meetings where short-term planning was discussed for the answering of individual research questions. The author was responsible for meeting the milestones in a timely fashion.

  Each publication in the SNMP field was preceded by a general plan, consisting of items indicating what was supposed to be completed at specific dates. This was supplemented by periodic meetings discussing draft versions of the papers with the involved authors.

- **Adjust problem statement and work schedule in accordance with interim evaluations**
  Regular meetings with the graduation committee have resulted in intermediate changes to both the planning and the research questions related to the problem statement. These changes were usually the result of newly obtained knowledge following interim research results.

  The work at SURFnet in the field of DNS has seen changes to the work plan. For instance, the work plan originally involved a large set of possible solutions that could have resulted in a more superficial set of results. Instead, the actual results are limited to a portion of the original work plan, but with more detail and testing than originally planned. These alterations have let to the results presented in the thesis.

  Besides that, there were also adjustments to the problem statement and the work schedule for the SNMP publications, where regular meetings resulted in mostly minor changes. However, personal circumstances of the author required a significant alteration to the work plan, in order to cope with the situation at that time.

- **Analyze different possible solutions and motivate a choice between them**
  Defining a problem statement goes hand in hand with the orientation of possible solutions to the problem at hand. A proper understanding of related

work is required to obtain a view of possible solutions that still need to be explored, but also of those that have already been considered by others. The DNS research and its work plan consist of multiple solutions, of which two were actually explored following intermediate alterations to the work plan. Different solutions have been considered for the publications in the SNMP field.

- **Communicate the research and design activities both written and in presentations**
  The research results in both the DNS and the SNMP field have been published in an IETF Internet-Draft, a paper and a poster. The results have also been presented in various settings, including an IETF meeting, an AIMS conference and a session during a TERENA Network Conference.

- **Show the ability of reflection on the problem, on the research/design approach, on the solution and on ones own performance**
  Preliminary results in both the DNS and SNMP research have led to alterations to the proposed solutions considered at that time. For instance, intermediate research results resulted in changes to the tools under development. For the author, this research was also a moment of improvement of his own performance, by means of specific and constructive feedback regarding presentation, writing style and more.

- **Demonstrate creativity and the ability to work independently**
  New information during a research phase can lead to changes to many aspects of the research itself. This has affected the planning, the work plan more generally, but also the shaping of actual solutions. These changes require creativity that is directed by knowledge of the field following a literature study, but also as a result of many discussions with external experts (e.g. NLnet Labs, TNO). The author was able to work in an independent manner while writing publications, presenting results and during the research phase.

1.2 Research Topics

This thesis covers the work in two fields: DNS and SNMP. The DNS research and the SNMP research have a common background due to the analysis of real-world traces. The publications related to DNS are the result of a research period at SURFnet. The publications related to SNMP result from research following the final research assignment for the author's Bachelor of Science degree. The following list discusses the various topics in each of the two fields that have been covered by this thesis and its publications.

- **The Domain Name System (DNS)**
  - The DNS research at SURFnet involves work related to the problem with fragmented DNS response messages that may result in unreachable DNS zones for some resolvers. This is a problem that becomes worse when DNSSEC is more widely deployed. The final results of the DNS research, an extensive description of real-world analyses of DNS traffic and two solutions to the problem are presented in a paper that is to be accepted for publication in the IEEE Network Magazine.
  - Preliminary results of this research were presented with potential solutions at the TERENA Networking Conference (TNC) in the form of a poster and a presentation during a session on middleware.
- **The Simple Network Management Protocol (SNMP)**
  - The first publication regarding SNMP is an IETF Internet-Draft presenting definitions for the analyses of SNMP network traces. These definitions allow for specific separation of traces into smaller sets of related SNMP messages. These definitions form a basis for other research related to SNMP traffic.
  - The aforementioned IETF Internet-Draft did not pass the draft state, because of a lack of commitment and interest from other participants in the NMRG group of the IETF. A modified version of the draft resulted in a publication at AIMS.
  - A paper using the SNMP trace analyses definitions is the last publication of this thesis. This paper discusses separating periodic from aperiodic SNMP traffic. The publication of this paper is pending.

1.3 Thesis Organization

This thesis is organised as follows:

1.3.1 DNS

Publications and pending publications regarding the research assignment at SURFnet involving DNS:

- **Chapter 2** Draft paper containing the final research results. This paper is to be submitted to IEEE Network Magazine (2012).
- **Chapter 3** Poster with preliminary findings and two solutions to the problem of the research performed at SURFnet. Poster presented at the TERENA Networking Conference (TNC, 2012), Reykjavik, Iceland.

1.3.2 SNMP

Publications and pending publications regarding the research in the SNMP field:

- **Chapter 4** Informational Internet-Draft published and presented at 71st IETF meeting (2008), Philadelphia, United States.
- **Chapter 5** Paper published in the proceedings of the 2nd International Conference on Autonomous Infrastructure, Management and Security (AIMS, 2009), Bremen, Germany.
- **Chapter 6** Draft paper on separating periodic from aperiodic SNMP traffic. Submission target and time to be decided (2010).

Chapter 2

The text of chapter 2 is basically an article that is currently under review for IEEE Communications Magazine (ComMag). Articles submitted to ComMag may not have been published before; therefore the text of chapter 2 has been removed from this document. Interested readers can obtain a copy of the submitted email by sending an email to Gijs van den Broek or Aiko Pras.

Chapter 3

TENERA Network Conference Poster - Improving Response Deliverability in DNSSEC

3.1 Poster

A poster containing results following real-world trace analysis of DNS traffic and two solutions to improve DNS response deliverability has been presented at the TERENA Networking Conference (TNC) in May 2012. The provided solutions answer the main research question for the DNS research part of this thesis with respect to how we can provide DNS responses to resolvers that are not able to receive fragmented DNS response messages from authoritative name servers.

Improving Response Deliverability in DNS(SEC)

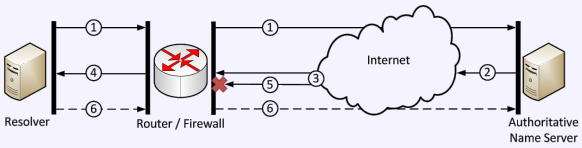
Gijs van den Broek[*‡], Roland van Rijswijk[‡], Aiko Pras*, Anna Sperotto*

***University of Twente, Enschede, The Netherlands (j.g.vandenbroek@student.utwente.nl, a.pras@utwente.nl, a.sperotto@utwente.nl)**

[‡]SURFnet bv, Utrecht, The Netherlands (roland.vanrijswijk@surfnet.nl)

The Domain Name System provides a critical service on the Internet, where it allows host names to be translated to IP addresses. However, it does not provide any guarantees about authenticity and origin integrity of resolution data. DNSSEC attempts to solve this through the application of cryptographic signatures to DNS records. These signatures generally result in larger responses compared to plain DNS responses. Some of these larger responses experience fragmentation, which in turn might be partially blocked by some firewalls. Apparently unresolvable zones may in those cases be a consequence. Analysis of DNS traffic suggests that at least one per cent of all resolvers experience this problem with our signed zones. However, we suspect this number to be much larger. In our presentation we will elaborate on the potential extent of this problem and propose to test two solutions. We intent to test both solutions in our production environment.

Problem Overview



	Description
1	DNS query sent to authoritative name server
2	DNS response returned
3	DNS response fragmented into IP fragments due to lower MTU
4	First IP fragment of DNS response arrives at resolver
5	Second IP fragment of DNS response is blocked at firewall
6	An ICMP Fragment Reassembly Time Exceeded message is sent 30 seconds later

Table 1: Problem resolver not able to obtain a fragmented DNS response.

DNS Traffic Analysis

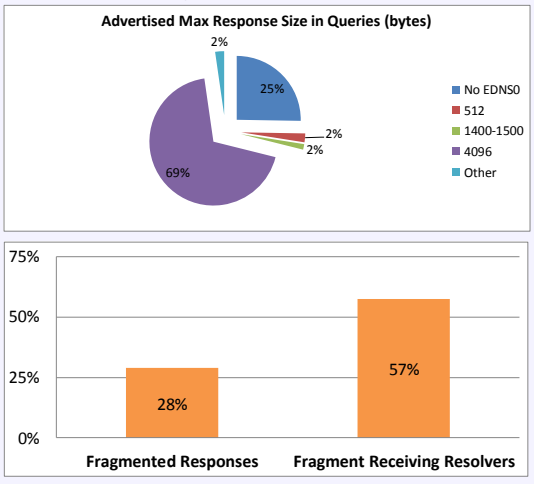


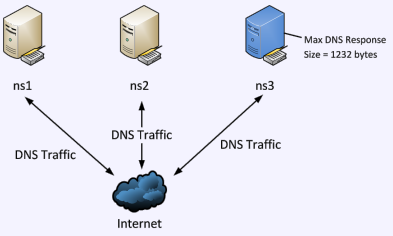
Figure 1, 2: Distribution of max. DNS response size in queries, percentage of all UDP DNS responses being fragmented and percentage of all resolvers receiving fragments (at ns3.surfnet.nl, sample 8.4 mln. messages).

Resolver Behavioural Characteristics	Unique Resolvers
Case 1: Sending ICMP Fragment Reassembly Time Exceeded (FRTE)	1.1%
Case 2: Removal of EDNS0 header in retries	2.4%
Case 3: Retries for large (>512 bytes) responses exceed 4%	9.7%
Case 4: Reduced advertised buffer size in retries	3.5%
Case 5: TCP fallback w/o truncated UDP response preceding it	<0.1%

Table 2: Resolver characteristics detected at ns1, ns2 and ns3.surfnet.nl. This involved 386,648 unique IPv4 and IPv6 resolvers and 40.5 mln messages.

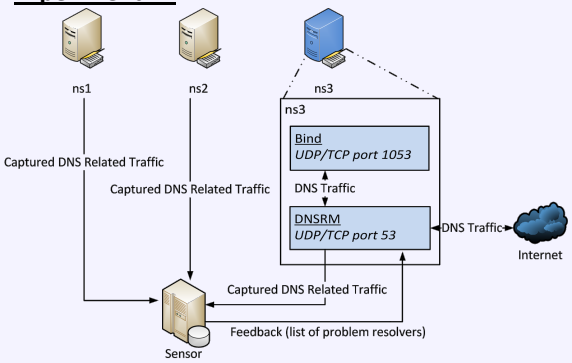
Experiment #1

We will use three authoritative name servers part of the surfnet.nl zone for both experiments. All of these name servers have the same configuration, unless stated otherwise.



- Name server ns3 returns responses with a maximum size of: *min*('max. size advertised in query', 1232) bytes.

Experiment #2



- Name server ns3 runs DNSRM on port 53, which forwards inbound DNS traffic to a local Bind process on port 1053. It also sends a copy of the live traffic feed to the sensor.
- Name servers ns1 and ns2 merely forward a copy of their live traffic feed to the sensor to improve the detection of problem resolvers.
- The sensor analyses this traffic in order to determine problem resolvers. It returns a list of detected problem resolvers to the DNSRM process on ns3.
- DNSRM alters advertised maximum response size advertisements for problem resolvers only to: *min*('max. size advertised in query', 'detected maximum for this resolver').

Comparison

Experiment 1 affects every resolver querying authoritative name server ns3, while experiment 2 involves the detection of problem resolvers and manipulating only those queries from such problem resolvers. Experiment 1 is very simple compared to the elaborate setup required for experiment 2, but it may be considered bad netizenship. It is questionable if the results of experiment 2 outweigh the results and simplicity of experiment 1.

SURF NET

Presentation Wednesday 23/05
14:00 - 15:30 Hall 3

UNIVERSITY OF TWENTE.

FIGURE 3.1: TERENA Networking Conference 2012 - Poster

Chapter 4

IETF Internet-Draft: SNMP Trace Analysis Definitions

Authors: J.G. van den Broek (University of Twente), J. Schönwälder (Jacobs University Bremen), A. Pras (University of Twente), M. Harvan (ETH Zurich)

This document originally appeared as an IETF Informational Internet-Draft (*draft-schoenw-nmrg-snmp-trace-definitions-02*). This document has been edited for style.

4.1 Abstract

The Network Management Research Group (NMRG) started an activity to collect traces of the Simple Network Management Protocol (SNMP) from operational networks. To analyze these traces, it is necessary to split potentially large traces into more manageable pieces that make it easier to deal with large data sets and simplify the analysis of the data.

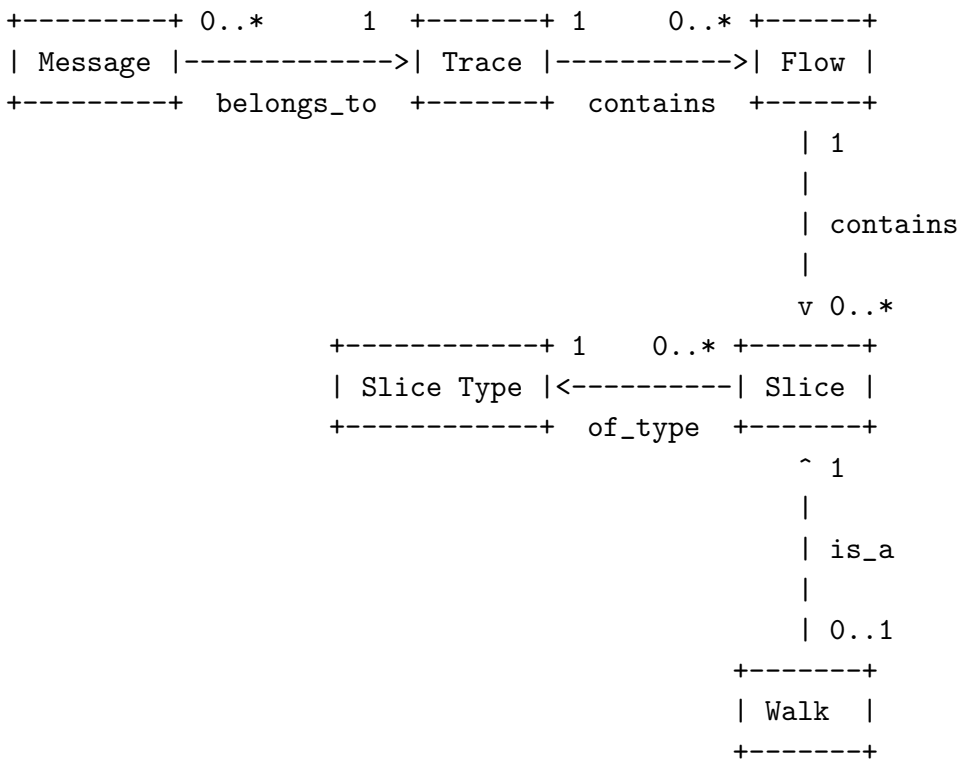
This document provides some common definitions that have been found useful for implementing tools to support trace analysis. This document mainly serves as a reference for the definitions underlying these tools and it is not meant to explain all the motivation and reasoning behind the definitions. Some of this background information can be found in other research papers.

4.2 Introduction

The Simple Network Management Protocol (SMMP) was introduced in the late 1980s. Since then, several evolutionary protocol changes have taken place, resulting in the SNMP version 3 framework (SNMPv3) [RFC3410][RFC3411], published as full standard in 2002. Extensive use of SNMP has led to significant practical experience by both network operators and researchers. However, up until now only little research has been done on characterizing and modeling SNMP traffic.

Since recently, network researchers are in the possession of network traces, including SNMP traces captured on operational networks. The availability of SNMP traces enables research on characterizing and modeling real world SNMP traffic. However, experience with SNMP traces has shown that the traces must be large enough in order to make proper observations. A more detailed motivation for collecting SNMP traces and guidelines on how to capture SNMP traces can be found in [ID-IRTF-NMRG-SNMP-MEASURE].

Unfortunately, the analysis of large SNMP traces can take a large amount of processing time. Therefore, it is often desirable to focus the analysis on smaller, relevant sections of a trace. This in turn requires a proper way to identify these smaller sections of a trace. This document describes a number of identifiable sections within a trace which make specific research on these smaller sections more practical. The following figure shows the various entities associated with SNMP traces and how they relate to each other.



This document defines the various entities (boxes) shown in the above figure. These definitions can be implemented by tools that can split SNMP traces into smaller sections for further analysis.

The most central entity in the figure above is an SNMP trace, consisting of a potentially large set of SNMP messages. An SNMP trace is the result of recording SNMP traffic on a specific network for a specific time duration. Such a trace may, depending on the number of hosts in the respective network, contain SNMP messages exchanged between possibly many different SNMP engines. The messages contained in a trace may be represented in different formats. For the purpose of this document, the simple comma separated values (CSV) format defined in [ID-IRTF-NMRG-SNMP-MEASURE] contains sufficient information to split a trace into smaller sections.

The SNMP messages belonging to an SNMP trace may have been exchanged between many different SNMP engines running on different hosts. Therefore, a first obvious way of separating a trace into smaller sets of SNMP messages is the separation of a trace into flows. Each flow contains only those SNMP messages of an SNMP trace that have been exchanged between two network layer endpoints. Such a separation may be necessary in case one wants to analyze specific SNMP traffic characteristics (e.g., number of agents managed by a management station) and wants to rule out network endpoint specific behaviour (e.g., different SNMP management stations may have different polling configurations).

Flows within traces can still be quite large in terms of the number of messages they contain. Therefore, it may be necessary to split a flow into even smaller sections called slices. A slice contains all SNMP messages of a given flow that are related to each other in time and referenced information. Splitting a flow into slices makes it possible to separate SNMP messages within traces that belong to each other.

For example, a slice may contain the SNMP messages exchanged between an agent and a manager, which polls that agent in a single polling instance. The manager may be configured to poll that agent every once in a while. If the requested information from the agent remains unchanged, then the respective slices of SNMP traffic occurring between this manager and agent will be highly comparable. In such a case the slices will be of the same slice type. Similar slices will thus be considered of the same slice type and incomparable slices will not be of the same slice type.

Besides the fact that each slice is of specific slice type, slices can also be of a specific form with respect to the messages encompassing a slice. For example, slices containing a sequence of linked GetNext or GetBulk requests are commonly called an SNMP walk. Note that only a subset of all slices will be walks.

4.3 Messages

SNMP messages carry PDUs associated with well defined specific protocol operations [RFC3416]. The PDUs can be used to classify SNMP messages. Following are a number of definitions that help to classify SNMP messages based on the PDU contained in them. These definitions will be used later on in this document.

Notation The properties of an SNMP message M are denoted as follows:

M.type = operation type of message M (get, getnext, ...)
M.class = class of message M (according to RFC 3411)
M.tsrc = transport layer source endpoint of message M
M.tdst = transport layer destination endpoint of message M
M.nsrc = network layer source endpoint of message M
M.ndst = network layer destination endpoint of message M
M.reqid = request identifier of message M
M.time = capture timestamp of message M
M.oids = OIDs listed in varbind list of message M
M.values = values listed in varbind list of message M

These properties of an SNMP message can be easily extracted from the exchange formats defined in [ID-IRTF-NMRG-SNMP-MEASURE].

Definition *read request message*: A read request message is a message M containing a PDU of type GetRequest, GetNextRequest, or GetBulkRequest.

Definition *write request message*: A write request message is a message M containing a PDU of type SetRequest.

Definition *notification request message*: A notification request message is a message M containing a PDU of type InformRequest.

Definition *notification message*: A notification message is a message M containing a PDU of type Trap or InformRequest.

Definition *request message*: A request message is a message M which is either a read request message, a write request message, or a notification request message.

Definition *response message*: A response message is a message M containing a PDU of type Response or of type Report.

Report messages are treated like Response messages since the SNMPv3 specifications currently use Report messages only as an error reporting mechanism, always triggered by the processing of some request messages. In case future SNMP versions or extensions use Report messages without having a request triggering the generation of Report messages, the definition above may need to be revisited.

Definition *non-response message*: A non-response message is a message M which is either a read request message, a write request message, or a notification message.

Definition *command message*: A command message is a message M which is either a read request message or a write request message.

Definition *command group messages*: A set of command group messages consists of all messages M satisfying either of the following two conditions:

(C1) M is a command message
(C2) M is a response message and there exists a command message C such that the following holds:

M.reqid = C.reqid
M.tdst = C.tsrc
M.tsrc = C.tdst
(M.time - C.time) $< t$

The parameter t defines a maximum timeout for response messages.

This definition requires that the response message originates from the transport endpoint over which the request message has been received. This is not strictly required by SNMP transport mappings and in particular the UDP transport mapping allows to send responses from different transport endpoints. While sending response messages from a different transport endpoint is legal, it is also considered bad practice causing interoperability problems, since some management systems do not accept such messages.

It was decided to require matching transport endpoints since doing so significantly simplifies the procedures below and avoids accidentally confusing requests and responses. Implementations responding from different transport endpoints will lead to (a) a larger number of requests without related responses (and likely no retries) and (b) a similarly large number of responses without a matching request. If such behavior can be detected, the traces should be investigated and if needed the transport endpoints corrected. The requirement for matching transport endpoints only affects request / response pairs. It is perfectly fine for a manager to use different transport layer endpoints in different polling instances or even for different operations (i.e., slices) within the same polling instance.

Definition *notification group messages*: A set of notification group messages consists of all messages M satisfying either of the following two conditions:

(N1) M is a notification message
(N2) M is a response message and there exists a notification request message

N such that the following holds:

M.reqid = N.reqid
M.tdst = N.tsrc
M.tsrc = N.tdst
(M.time - N.time) $< t$

The parameter t defines a maximum timeout for response messages.

This definition again requires matching transport endpoints for notification group messages.

4.4 Traces

Traces are (large) sets of SNMP messages that are the result of recording SNMP traffic using a single traffic recording unit (e.g., using tcpdump) on a network segment carrying traffic of one or more managers and agents. Traces being used in the remainder of this document may be altered as a result of anonymization, which may result in some message information loss.

Definition *trace*: An SNMP trace (or short "trace") T is an ordered set of zero or more SNMP messages M. All messages M in T are chronologically ordered according to the capture timestamp M.time.

4.5 Flows

Traces may contain SNMP messages that have been exchanged between possibly many different network layer endpoints. One way of making an initial separation of such a trace into more manageable pieces is by splitting the messages into flows. Each flow contains only messages that have occurred between two network layer endpoints.

Definition *flow*: A flow F with parameter t is the set of messages of an SNMP trace T with the following properties:

(F1) All response messages originate from a single network endpoint.
(F2) All non-response messages originate from a single network endpoint.
(F3) All messages are either command group messages with parameter t or notification group messages with parameter t.

Parameter t defines the maximum timeout for response messages and is the same for both the notification group messages and the command group messages which a flow may consist of. Parameter t is the same for both cases, because they each may contain non-response messages that originate from a single network endpoint with one timeout characteristic. The value of t should be chosen such that only response messages to the respective non-response messages are considered part of the same flow. Analysis of a large number of traces shows that 25 seconds is a proper default value for t.

It is possible that response messages of a trace cannot be classified to belong to any flow. This can happen if request messages triggering the response messages were not recorded (for example due to asymmetric routing), or because response messages were originating from transport endpoints different from the endpoint used to receive the associated request message.

Subsequently, flows containing only command group messages are called command flows. Similarly, flows containing only notification group messages are called notification flows.

This definition of a flow indicates that it can be either unidirectional (e.g., a manager sending non-response messages to a non-responding agent), or bidirectional (e.g., a manager reading a table from an agent). This is different from other flow definitions, like the NetFlow definition [RFC3954].

The flow definition is mostly consistent with the definition of an SNMP flow used in [SPHSM07]. The difference is that the tool used to generate the data reported in [SPHSM07] did only require that the network layer source endpoint of the response messages matches the destination network layer endpoint of the associated request messages.

Definition *flow initiator*: A flow initiator is the network layer endpoint of the two endpoints involved in a flow, which is responsible for sending the first non-response message.

Notation The properties of a flow F are denoted as follows:

F.type = type of the flow F (command/notification)
F.nsrc = network layer source endpoint of F
F.ndst = network layer destination endpoint of F
F.start = timestamp of the first message in F
F.end = timestamp of the last message in F
F.init = initiator of the flow F
F.t = parameter t of F
(maximum timeout for response messages)

Following is an example of a trace consisting of SNMP messages that were exchanged between different network layer endpoints. The network layer endpoints are represented by A, B, C and D. This trace will be separated into flows.

Message	Time [s]	Direction	PDU type	Req. ID.
0	0.0	A -> B	GetNext Request	1
1	0.04	C -> D	Get Request	10
2	0.05	B -> A	Response	1
3	0.08	D -> C	Response	10
4	0.11	A -> B	GetNext Request	2
5	0.15	B -> A	Response	2
6	0.18	A -> B	GetNext Request	3
7	0.22	D -> C	Trap	14
8	0.25	B -> A	Response	3

This trace contains SNMP messages that have been exchanged between different network layer endpoints. One flow will consist of SNMP messages that have been exchanged between network layer endpoints A and B, where all response messages originate from B and all non-response messages originate from A.

Message	Time [s]	Direction	PDU type	Req. ID.
0	0.0	A -> B	GetNext Request	1
2	0.05	B -> A	Response	1
4	0.11	A -> B	GetNext Request	2
5	0.15	B -> A	Response	2
6	0.18	A -> B	GetNext Request	3
8	0.25	B -> A	Response	3

The minimum value of parameter t of this flow is 0.07 seconds, since that is the longest time between a request and its subsequent response message.

The second flow contains SNMP messages exchanged between network layer endpoints C and D, where all response messages originate from D and all non-response messages originate from C.

Message	Time [s]	Direction	PDU type	Req. ID.
1	0.04	C -> D	Get Request	10
3	0.08	D -> C	Response	10

The minimum value of parameter t for this flow is 0.04 seconds. The third and last flow contains the remaining SNMP messages of the trace that occurred between network layer endpoints C and D. In this case the non-response message originates from D.

Message	Time [s]	Direction	PDU type	Req. ID.
7	0.22	D -> C	Trap	14

There is no parameter t applicable to this flow, because there are no response messages.

4.6 Slices

Flows can still contain a large amount of SNMP messages. A flow should therefore be split up into even smaller sets of messages. One way of identifying meaningful subsets of messages of a flow would be by considering the behavior of managers and agents. In the case of managers, they are usually configured to perform regular polling instances. In such a polling instance, the manager might poll a number of agents. Since a flow contains only the messages exchanged between two network layer endpoints, a flow therefore probably consists of only a subset of the messages that are part of a polling instance. So, one option of finding smaller, meaningful subsets of messages within flows, would be by looking for messages that belong to a particular operation of a polling instance. Such a smaller set of messages is called a slice.

Definition *slice*: A slice S with parameter e is a subset of SNMP messages in a flow F for which the following properties hold:

(S1) All messages are exchanged between the same two transport endpoints (a single transport endpoint pair).

(S2) All non-response messages must have a PDU of the same type.

(S3) All messages with a PDU of type Get, Set, Trap, or Inform must contain the same set of OIDs.

(S4) Each GetNext or GetBulk message must either contain the same set of OIDs as the preceding request or it must be linked to a response of the last previously answered request (i.e., the request must contain at least one OID that has been contained in the (repeater) varbind list of a preceding response message of the last answered request message).

(S5) All Response messages must follow a previous request message.

(S6) For any two subsequent non-response messages Q1 and Q2 with Q1.time < Q2.time, the following condition must hold:

$$(\text{Q2.time - Q1.time}) < e$$

S1 requires that the messages of a single slice are exchanged between a single transport layer endpoint pair. This is different from the flow definition, which requires a single network layer endpoint pair. The choice of looking at the transport layer endpoints in the case of slices is based on the assumption that, for instance, multiple managers and agents might be operating from the same respective network layer endpoint. Another assumption is that a manager and an agent will only use a single transport layer endpoint respectively when they communicate for the duration of a slice. A previous section already mentioned some issues when a manager or agent uses different transport layer endpoints within a single slice.

The parameter e in S6 defines the maximum time between two non-response messages that belong to a slice. This parameter should be chosen such that unrelated non-response messages within a flow are not considered to be of the same slice (i.e., it is used to detect the end of a slice). Unrelated non-response messages are those that, for instance, belong to different polling instances. The parameter e should therefore be larger than the retransmission interval in order to keep retransmissions within a slice and smaller than the polling interval used by the slice initiator.

The value of parameter e might be closely related to parameter t of the respective flow the slice is part of. For instance, if parameter e is very large, than t is likely to be also very large and vice versa. Also, if parameter e is very small, than t is probably also very small. However, it is not possible to strictly state that e and t are always closely related to each other, because parameter e is specific for a

slice. This is in contrast with parameter t which is specific for a much larger set of messages, a flow.

Definition *slice initiator*: A slice initiator is one of the two transport layer endpoints involved in a slice, which is responsible for sending the chronologically first non-response message.

Notation The properties of a slice S are denoted as follows:

S.type	= type of non-response messages in S
S.tsrc	= transport layer endpoint of initiator of S
S.tdst	= transport layer endpoint of non-initiator of S
S.start	= timestamp of the chronologically first message in S
S.end	= timestamp of the chronologically last message in S
S.init	= initiator of slice S
S.e	= parameter e of S (maximum time between two non-response messages)

Following is an example of a flow F consisting of SNMP messages that have been exchange place between transport layer endpoints A, B, C and D. Note that even though there are multiple transport layer endpoints, it still means that within this single flow only two network layer endpoints can be identified.

Message	Time [s]	Direction	PDU type	Req. ID	OIDs
0	0.0	A -> B	GetNext Request	1	alpha
1	0.05	B -> A	Response	1	alpha.1
2	0.11	A -> B	GetNext Request	2	alpha.1
3	0.15	B -> A	Response	2	alpha.2
4	0.18	A -> B	GetNext Request	3	alpha.2
5	0.25	B -> A	Response	3	beta.1
6	300.0	C -> D	GetNext Request	4	alpha
7	300.06	D -> C	Response	4	alpha.1
8	300.14	C -> D	GetNext Request	5	alpha.1
9	300.21	D -> C	Response	5	alpha.2
10	300.28	C -> D	GetNext Request	6	alpha.2
11	300.32	D -> C	Response	6	beta.1

This flow contains SNMP messages that have been exchanged between two different pairs of transport layer endpoints. According to S1, this means that the set of messages in this flow can be separated into at least two sets of messages that might be slices:

Message	Time [s]	Direction	PDU type	Req. ID	OIDs
0	0.0	A -> B	GetNext Request	1	alpha
1	0.05	B -> A	Response	1	alpha.1
2	0.11	A -> B	GetNext Request	2	alpha.1
3	0.15	B -> A	Response	2	alpha.2
4	0.18	A -> B	GetNext Request	3	alpha.2
5	0.25	B -> A	Response	3	beta.1

Message	Time [s]	Direction	PDU type	Req. ID	OIDs
6	300.0	C -> D	GetNext Request	4	alpha
7	300.06	D -> C	Response	4	alpha.1
8	300.14	C -> D	GetNext Request	5	alpha.1
9	300.21	D -> C	Response	5	alpha.2
10	300.28	C -> D	GetNext Request	6	alpha.2
11	300.32	D -> C	Response	6	beta.1

S2 does not result in a further separation of these sets into smaller sets of SNMP messages that might be slices. This, because all the listed non-response messages are of the same PDU type. Neither is S3 affecting the current sets.

S4 also does not result in a further subdivision of these two sets, because in each of the two sets the non-response messages, that have preceding response messages, have the same OID in their varbind list as the respective preceding response. For example, the second and third non-response message in both sets have the same OID compared to the respective preceding response message (i.e., these non-response messages are a part of a single polling instance).

S5 also does not affect these two sets, because both sets already contain only response messages that are the result of a non-response message that occurred before each response message respectively (i.e., each response has a request identifier that is the same as the request identifier of a non-response message listed before that response message).

S6 states that parameter e should be chosen such that unrelated requests within a flow are not considered to be of the same slice. Considering the given flow and its messages, a proper value for e should be $0.14 \leq e < 299.82$ seconds. Such a value for parameter e will separate the flow into two apparent polling instances, which each contain the same set of messages as those two listed above.

As a result, the two listed sets are also slices. The given flow thus consists of the following two slices with the value of parameter e as defined above.

Following is an example of a flow F consisting of SNMP messages that have been exchanged between transport layer endpoints A, B and C. Note again that even though there are multiple transport layer endpoints, it still means that within this single flow only two network layer endpoints can be identified.

Message	Time [s]	Direction	PDU type	Req. ID	OIDs
0	0.0	A -> B	Get Request	1	alpha.1
1	0.06	B -> A	Response	1	alpha.1
2	0.12	A -> B	Get Request	2	beta.1
3	0.17	B -> A	Response	2	beta.1
4	300.0	A -> B	Get Request	3	alpha.1
5	300.05	B -> C	Set	10	gamma.1
6	300.07	B -> A	Response	3	alpha.1
7	300.14	A -> B	Get Request	4	beta.1
8	300.19	B -> A	Response	4	beta.1

This flow contains messages that are exchanged between different pairs of transport layer endpoints. Two different pairs of transport layer endpoints are used within this flow. According to S1, this means that at least two sets of messages might be slices:

Message	Time [s]	Direction	PDU type	Req. ID	OIDs
0	0.0	A -> B	Get Request	1	alpha.1
1	0.06	B -> A	Response	1	alpha.1
2	0.12	A -> B	Get Request	2	beta.1
3	0.17	B -> A	Response	2	beta.1
4	300.0	A -> B	Get Request	3	alpha.1
6	300.07	B -> A	Response	3	alpha.1
7	300.14	A -> B	Get Request	4	beta.1
8	300.19	B -> A	Response	4	beta.1

Message	Time [s]	Direction	PDU type	Req. ID	OIDs
5	300.05	B -> C	Set	10	gamma.1

Applying S2 to each of the two sets does not result in more sets, because each set only contains non-response messages of one PDU type respectively.

S3 requires separating the first set of SNMP messages into two sets, because the Get Request messages must have the same set of OIDs. This results in the following three sets of SNMP messages.

Message	Time [s]	Direction	PDU type	Req. ID	OIDs
0	0.0	A -> B	Get Request	1	alpha.1
1	0.06	B -> A	Response	1	alpha.1
4	300.0	A -> B	Get Request	3	alpha.1
6	300.07	B -> A	Response	3	alpha.1

Message	Time [s]	Direction	PDU type	Req. ID	OIDs
2	0.12	A -> B	Get Request	2	beta.1
3	0.17	B -> A	Response	2	beta.1
7	300.14	A -> B	Get Request	4	beta.1
8	300.19	B -> A	Response	4	beta.1

Message	Time [s]	Direction	PDU type	Req. ID	OIDs
5	300.05	B -> C	Set	10	gamma.1

S4 is not applicable to either of these sets.

S5 does not affect these sets, because these sets already contain only response messages that are the result of a non-response message that occurred before each response message respectively.

Finally, S6 states that parameter e should be chosen such that unrelated requests within a flow are not considered to be of the same slice. The messages in the given flow suggest that two polling instances might have occurred. Therefore, an appropriate value for parameter e would be < 300 seconds. The determination of parameter e results in the separation of the first two sets into two sets each. Following are the resulting slices.

Message	Time [s]	Direction	PDU type	Req. ID	OIDs
0	0.0	A -> B	Get Request	1	alpha.1
1	0.06	B -> A	Response	1	alpha.1

Message	Time [s]	Direction	PDU type	Req. ID	OIDs
2	0.12	A -> B	Get Request	2	beta.1
3	0.17	B -> A	Response	2	beta.1

Message	Time [s]	Direction	PDU type	Req. ID	OIDs
4	300.0	A -> B	Get Request	3	alpha.1
6	300.07	B -> A	Response	3	alpha.1

Message	Time [s]	Direction	PDU type	Req. ID	OIDs
7	300.14	A -> B	Get Request	4	beta.1
8	300.19	B -> A	Response	4	beta.1

Message	Time [s]	Direction	PDU type	Req. ID	OIDs
5	300.05	B -> C	Set	10	gamma.1

4.7 Slice Prefix

As noted in the beginning of this document, it is desirable that slices can be tested for equality/comparability. This is where the slice prefix comes in. The slice prefix provides one of the means to compare slices. Using the slice prefix and a few other parameters (which will be discussed later on in this document) of a number of slices, one can determine which slices should be considered "equal" and which of them are incomparable. This will assist in the process of finding potentially other relations.

The slice prefix is a set of OIDs. This set is constructed from the messages that make up a single slice. So, for example, a slice that is the result of a manager requesting the contents of a particular table (with OID alpha) on an agent using a simple single varbind GetNext walk, starting at the table OID alpha, shall yield a slice prefix which consists of the OID alpha.

Because the aim is to compare various slices using the slice prefix (along some other characteristics of a slice), this implicitly suggests the need to know whether a number of slices are the result of the same behaviour (i.e., specific configuration) of the initiating party of these slices. For example, one may want to know whether a number of slices that involve a single manager and a single agent were the result of just one specific configuration of that manager. Multiple slices, that may all be initiated by that same manager and each slice possibly occurred in different polling instances, may in fact be the result of the same specific configuration of that particular manager. So, since in this case the specific configuration of the manager is only relevant for determining the behaviour, the slice prefix should be constructed based on OIDs in messages originating from that manager only. More generally, only the messages within slices that are sent by the initiating party (the non-response messages) are considered for the determination of the respective slice prefix of a slice.

The resulting set of OID prefixes will represent the behaviour of the respective initiating party of that slice. This allows us to compare different slices.

Following is a short introductory example which depicts what a slice could consist of and how one could determine the slice prefix in such a general case.

Consider the case of a single manager A polling a specific agent B. More specifically, the manager A is configured to retrieve the complete contents of two columns alpha and beta of a some table. The resulting slice may contain the following messages:

```
-------------------------------------------------------------------
Message | Direction | PDU type        | OIDs
-------------------------------------------------------------------
   0        A -> B    GetNext Request   alpha, beta
```

```
1        B -> A    Response          alpha.1, beta.1
2        A -> B    GetNext Request   alpha.1, beta.1
3        B -> A    Response          alpha.2, beta.2
4        A -> B    GetNext Request   alpha.2, beta.2
5        B -> A    Response          gamma.1, delta.1
-----------------------------------------------------------------
```

The manager starts with a GetNext request referencing two OIDs, alpha and beta. The agent B replies in message 1 with the first items of each of the referenced columns. The manager in turn goes on obtaining data from these two columns until it receives message 5, which indicates that the manager has received all of the data from the two columns.

It can be easily concluded that the manager was configured to retrieve the contents of the two columns alpha and beta (the slice prefix). A different slice involving the same manager and agent and that is again the result of the same configuration of the manager, should be considered "equal" to this one because the two slices are the result of the same behaviour. It should however be mentioned that such a second slice might contain a different number of messages, since the contents of the tables on the agent side might have changed over time. This underlines the previously made remark that only the messages originating from the initiating party should be considered in this process, because they will (in such a scenario) always illustrate the same behaviour of the initiating party.

The previous example now makes it possible to give a more formal definition of a slice prefix. This is accomplished by first defining a slice signature which then leads to the definition of a slice prefix.

Definition *slice signature*: A slice signature S.sig of a slice S is a set of OIDs derived from the OIDs contained in the non-response messages of a slice. Let r(S) denote the set of response messages of slice S and n(S) the set of non-response messages of S. Then the set S.sig consists of the following OIDs:

```
case S.type = GetNext or S.type = GetBulk:

     S.sig =      U      (n.oids)  -      U      (r.oids)
               n in n(S)                r in r(S)

otherwise:

     S.sig =      U      (n.oids)
               n in n(S)
```

Definition *OID prefix*: An OID a = a_1.a_2...a_n is a prefix of OID b = b_1.b_2...b_m if and only if

(P1) n < m and

(P2) a_i = b_i for $1 \leq i \leq n$.

Definition (slice prefix): The slice prefix S.slice is the set of all OIDs o in S.sig for which there is no p in S.sig such that p is a prefix of o.

Following is an example to illustrate the definitions just described: Consider the case that a single manager A polling an agent B. More specifically, the manager A is programmed to retrieve the complete contents of two single column tables alpha and beta. Besides that, the manager now also requests the sysUpTime in the first request the manager sends to B. A resulting slice may contain the following messages:

```
-------------------------------------------------------------------
Message | Direction | PDU type        | OIDs
-------------------------------------------------------------------
   0        A -> B    GetNext Request   sysUpTime, alpha, beta
   1        B -> A    Response          sysUpTime.0, alpha.1,beta.1
   2        A -> B    GetNext Request   alpha.1, beta.1
   3        B -> A    Response          alpha.2, beta.2
   4        A -> B    GetNext Request   alpha.2, beta.2
   5        B -> A    Response          gamma.1, delta.1
-------------------------------------------------------------------
```

The slice prefix is determined as follows:

```
1. The union of all OIDs in non-response messages is the following
   set:
          N = { sysUpTime, alpha, beta,
                alpha.1, beta.1, alpha.2, beta.2 }

2. The union of the OIDs in response messages is the following set:
          R = { sysUpTime.0, alpha.1, beta.1,
                alpha.2, beta.2, gamma.1, delta.1 }

3. Subtracting the two sets results in the slice signature S.sig:
          S.sig = N - R = { sysUpTime, alpha, beta }

4. Since none of the OIDs in S.sig is a prefix of any other OID in
   S.sig, the slice prefix is equal to the slice signature:
          S.prefix = { sysUpTime, alpha, beta }
```

This set of OIDs describes the behavior of the initiating party of this particular slice. Within the context of the described configuration of the initiating party is it apparent that a subsequent polling instance of the mentioned manager A polling agent B will result in the same slice signature.

Following is a more elaborate slice for which the slice prefix is determined. Consider again the case that a single manager A is set to poll a specific agent B. Manager A is programmed to retrieve some values from B. A single slice may contain the following messages:

Message	Direction	PDU type	OIDs
0	A -> B	GetNext Request	alpha, beta
1	B -> A	Response	alpha.1, beta.1
2	A -> B	GetNext Request	alpha.1, beta.1
3	B -> A	Response	alpha.2, beta.3
4	A -> B	GetNext Request	beta.2, alpha.2, sysUpTime
5	B -> A	Response	beta.3, alpha.3 sysUpTime.0
6	A -> B	GetNext Request	beta.3, alpha.3
7	B -> A	Response	gamma.1, alpha.4
8	A -> B	GetNext Response	alpha.4
9	B -> A	Response	delta.1

This slice has a number of interesting properties:

- Not all columns in the table have an equal length.
- The manager is set to request the sysUpTime on an irregular basis (i.e., every few requests).
- The manager attempts to fill "holes" in the table.
- The order of referenced OIDs in GetNext messages changes.

All of these properties do not influence the process for determining the slice signature. The slice prefix is constructed as follows:

1. The union of all OIDs in non-response messages is the following set:

```
      N = { alpha, beta, alpha.1, beta.1,
            beta.2, alpha.2, sysUpTime, beta.3, alpha.3, alpha.4 }
```

2. The union of the OIDs in response messages is the following set:

```
      R = { alpha.1, beta.1, alpha.2, beta.3, alpha.3,
            sysUpTime.0, gamma.1, alpha.4, delta.1 }
```

3. Subtracting the two sets results in the slice signature S.sig:

```
      S.sig = N - R = { alpha, beta, beta.2, sysUpTime }

      The element beta.2 exists, because the manager was trying to fill
      a "hole" in a table.  Since these holes reside in the tables on
      the agent side and may change dynamically, they do not really
      help in describing the behavior of the initiating party.
```

4. Since beta is a prefix of beta.2, the slice prefix becomes the following set:

```
      S.prefix = { alpha, beta, sysUpTime }

      The slice prefix does not include beta.2 anymore and thus a
      manager retrieving the same columns alpha and beta with and
      without holes will produce slices with the same slice prefix.
```

Finally, consider a slice where s.type is not GetBulk or GetNext. Manager A is programmed to poll some values from B. A single slice may contain the following messages:

Message	Direction	PDU type	OIDs
0	A -> B	Get Request	alpha.1, beta.1
1	B -> A	Response	alpha.1, beta.1

The slice prefix is determined as follows:

1. The slice signature is the union of all OIDs in non-response messages:

```
      S.sig = { alpha.1, beta.1 }
```

```
2. Since none of the OIDs in S.sig is a prefix of any other OID in
   S.sig, the slice prefix is equal to the slice signature:

       S.prefix = { alpha.1, beta.1 }
```

4.8 Slice Type

As described previously, the slice type allows for comparing slices. This means that any number of slices that are of the same slice type may be considered an equivalence class and may therefore be considered to be the result of the same behaviour of the slice initiator.

Definition *slice equivalence*: Two slices A and B satisfy the binary slice equivalence relation A $\sim$ B if the following properties hold:

(EQ1) All messages in A and B have been exchanged between the same two network layer endpoints.

(EQ2) All read request messages, write request messages, and notification messages in A and B originate from the same network layer endpoint.

(EQ3) All non-response messages in A and B are of the same type.

(EQ4) The slices A and B have the same prefix, that is A.prefix = B.prefix.

It can be easily seen that the relation $\sim$ is reflexive, symmetric, and transitive and thus forms an equivalence relation between slices.

Definition *slice type*: Let S be a set of slices, then all slices in the equivalence class
[A] = {s in S | s $\sim$ A} with A in S, are of the same slice type.

Following are two slices that are of the same equivalence class and are therefore of the same slice type.

The first slice contains messages that have been exchanged between transport layer endpoints A and B.

Message	Direction	PDU type	OIDs
0	A -> B	GetNext Request	alpha, beta
1	B -> A	Response	alpha.1, beta.1
2	A -> B	GetNext Request	alpha.1, beta.1
3	B -> A	Response	alpha.2, beta.2
4	A -> B	GetNext Request	alpha.2, beta.2
5	B -> A	Response	gamma.1, delta.1

The second slice contains messages that have been exchanged between transport layer endpoints C and D. However, the network layer endpoints of this slice are the same as the first slice and all non-response messages in both slices originate from the same network layer endpoint.

Message	Direction	PDU type	OIDs
0	C -> D	GetNext Request	alpha, beta
1	D -> C	Response	alpha.1, beta.1
2	C -> D	GetNext Request	alpha.1, beta.1
3	D -> C	Response	alpha.2, beta.2
4	C -> D	GetNext Request	alpha.2, beta.2
5	D -> C	Response	alpha.3, beta.3
6	C -> D	GetNext Request	alpha.3, beta.3
7	D -> C	Response	gamma.1, delta.1

EQ1 requires that both slices have occurred between the same two network layer endpoints, which is the case here.

EQ2 requires that all read request messages, write request messages, and notification messages (i.e., all non-response messages) originate from the same network layer endpoint, which is the case.

EQ3 is met, because all non-response messages are of PDU type GetNext Request in both cases.

Using the definition of a slice prefix as stated in the previous chapter, it can be determined that both slices have the prefix { alpha, beta }. As a result, also EQ4 is met.

Both slices are therefore part of the same equivalence class and are therefore of the same slice type.

4.9 Walks

Definition *walk*: A walk W is a slice S with the following properties:

(W1) The type S.type of the slice S is either GetNext request or GetBulk request.

(W2) At least one OID in the sequence of requests at the same varbind index must be increasing lexicographically while all OIDs at the same varbind index have to be non-decreasing.

Definition *strict walk*: A walk W is a strict walk if all OIDs in the sequence of requests at the same varbind index are strictly increasing lexicographically. Furthermore, the OIDs at the same index of a response and a subsequent request must be identical.

Definition *prefix constrained walk*: A walk W is a prefix constrained walk if all OIDs at the same index in the request have the same OID prefix. This prefix is established by the first request within the walk.

Following is an example of a slice that is also a walk.

Message	Direction	PDU type	OIDs
0	A -> B	GetNext Request	alpha, beta
1	B -> A	Response	alpha.1, beta.1
2	A -> B	GetNext Request	alpha.1, beta.1
3	B -> A	Response	alpha.2, beta.2
4	A -> B	GetNext Request	alpha.2, beta.2
5	B -> A	Response	gamma.1, delta.1

This slice is a walk, because it adheres to W1, since it contains only requests that are of PDU type GetNext Request. W2 is also met, because there are always two object identifiers at the same varbind index that are increasing lexicographically in the sequence of requests. Also, none of the object identifiers at the same varbind index is lexicographically decreasing.

This slice is also a strict walk, because all object identifiers in the sequence of requests at the same varbind index are strictly increasing lexicographically. Secondly, the object identifiers at the same index of a response and a subsequent request are always identical.

Finally, this walk is also a prefix constrained walk, because all object identifiers at the same index in the request have the same object identifier prefix. The first varbind index in the requests always has alpha as prefix and the second varbind index always has beta as prefix.

4.10 Concurrency

Definition *concurrency*: Two slices A and B of a given flow F are concurrent at time t if A.start $\leq t \leq$ A.end and B.start $\leq t \leq$ B.end.

Definition *flow concurrency level*: The concurrency level of a flow F at time t is given by the number of concurrent slices of F at time t.

Definition *slice initiator concurrency level*: The concurrency level of a slice initiator I at time t is given by the number of concurrent slices initiated by the slice initiator I at time t.

Definition *overlapping*: Two slices A and B of a given flow F are called overlapping if there exists a time t where A and B are concurrent.

Definition *delta time serial*: Two slices A and B of a given flow F with B.start $>$ A.end are called delta time serial if (B.start - A.end) $<$ delta.

4.11 Acknowledgements

This document was influenced by discussions within the Network Management Research Group (NMRG).

Part of this work was funded by the European Commission under grant FP6-2004-IST-4-EMANICS-026854-NOE.

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).

4.12 References

4.12.1 Normative References

[RFC3411] Harrington, D., Presuhn, R., and B. Wijnen, "An Architecture for Describing Simple Network Management

Protocol (SNMP) Management Frameworks", RFC 3411, December 2002.

[RFC3416] Presuhn, R., "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", RFC 3416, December 2002.

4.12.2 Informative References

[RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet Standard Management Framework", RFC 3410, December 2002.

[RFC3954] Claise, B., "Cisco Systems NetFlow Services Export Version 9", RFC 3954, October 2004.

[ID-IRTF-NMRG-SNMP-MEASURE] Schoenwaelder, J., "SNMP Traffic Measurements and Trace Exchange Formats", ID draft-irtf-nmrg-snmp-measure-03.txt, February 2008.

[SPHSM07] Schoenwaelder, J., Pras, A., Harvan, M., Schippers, J., and R. van de Meent, "SNMP Traffic Analysis: Approaches, Tools, and First Results", IFIP/IEEE Integrated Management IM 2007, May 2007.

4.13 Full Copyright Statement

WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

4.13.1 Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Chapter 5

AIMS Paper: SNMP Trace Analysis Definitions

Authors: J.G. van den Broek (University of Twente), J. Schönwälder (Jacobs University Bremen), A. Pras (University of Twente), M. Harvan (ETH Zurich)

This document originally appeared in the proceedings of the second AIMS conference. This document has been edited for style.

5.1 Abstract

The Network Management Research Group (NMRG) started an activity to collect traces of the Simple Network Management Protocol (SNMP) from operational networks. To analyze these traces, it is necessary to split potentially large traces into more manageable pieces that make it easier to deal with large data sets and simplify the analysis of the data. This document introduces some common definitions that have been found useful for implementing tools to support trace analysis.

5.2 Introduction

The Simple Network Management Protocol (SMMP) was introduced in the late 1980s. Since then, several evolutionary protocol changes have taken place, resulting in the SNMP version 3 framework (SNMPv3), published as full standard in 2002 [1, 2]. Extensive use of SNMP has led to significant practical experience by

both network operators and researchers. However, up until now only little research has been done on characterizing and modeling SNMP traffic.

Since recently, network researchers are in the possession of network traces, including SNMP traces, captured on operational networks. The availability of SNMP traces enables research on characterizing and modeling real world SNMP traffic. Experience with SNMP traces has shown that traces must be large enough in order to make proper observations. A more detailed motivation for collecting SNMP traces and guidelines how to capture SNMP traces can be found in [3].

The analysis of large SNMP traces can take a large amount of processing time. Therefore, it is often desirable to focus the analysis on smaller, relevant sections of a trace. This in turn requires a proper way to identify these smaller sections of a trace. This document describes a number of identifiable sections within a trace which make specific research on these smaller sections more practical.

The rest of the paper is structured as follows. An overview of the definitions is given in the next section and subsequent sections define messages, traces, flows, slices, slice prefixes, and slice types. Related and future work is discussed before the paper concludes.

5.3 Overview

Fig. 1 shows the various entities associated with an SNMP trace and how they relate to each other.

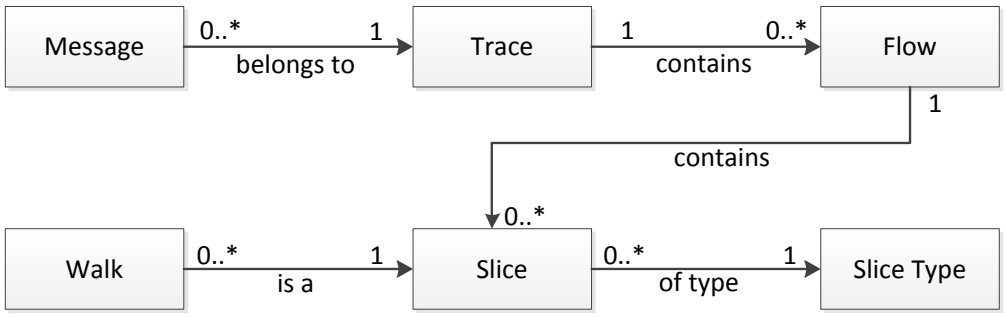


FIGURE 5.1: Relationship between messages, traces, flows, slices and slice types.

The most central entity in Fig. 1 is an SNMP trace, consisting of a potentially large set of SNMP messages. An SNMP trace is the result of recording SNMP traffic on a specific network for a specific time duration. Such a trace may, depending on

the number of hosts in the respective network, contain SNMP messages exchanged between possibly many different SNMP engines. The messages contained in a trace may be represented in different formats. For the purpose of this document, the simple comma separated values (CSV) format defined in [3] contains sufficient information to split a trace into smaller sections.

The SNMP messages belonging to an SNMP trace may have been exchanged between many different SNMP engines running on different hosts. Therefore, a first obvious way of separating a trace into smaller sets of SNMP messages is the separation of a trace into flows. Each flow contains only those SNMP messages of an SNMP trace that have been exchanged between two network layer endpoints. Such a separation may be necessary in case one wants to analyze specific SNMP traffic characteristics (e.g., number of agents managed by a management station) and wants to rule out network endpoint specific behaviour (e.g., different SNMP management stations may have different polling configurations).

Flows within traces can still be quite large in terms of the number of messages they contain. Therefore, it may be necessary to split a flow into even smaller sections called slices. A slice contains all SNMP messages of a given flow that are related to each other in time and referenced information. Splitting a flow into slices makes it possible to separate SNMP messages within traces that belong to each other.

For example, a slice may contain the SNMP messages exchanged between an agent and a manager, which polls that agent in a single polling instance. The manager may be configured to poll that agent every once in a while. If the requested information from the agent remains unchanged, then the respective slices of SNMP traffic occurring between this manager and agent will be highly comparable. In such a case the slices will be of the same slice type. Similar slices will thus be considered of the same slice type and incomparable slices will not be of the same slice type.

Besides the fact that each slice is of specific slice type, slices can also be of a specific form with respect to the messages encompassing a slice. For example, slices containing a sequence of linked GetNext or GetBulk requests are commonly called an SNMP walk. Note that only a subset of all slices will be walks.

5.4 Messages

SNMP messages carry protocol data units (PDUs) realizing a small set of well defined protocol operations [4]. The PDUs can be used to classify SNMP messages.

Notation 1. *The properties of an SNMP message M are denoted as follows:*

M.type = *operation type of message M (get, getnext, ...)*
M.class = *class of message M (according to RFC 3411)*
M.tsrc = *transport layer source endpoint of message M*
M.tdst = *transport layer destination endpoint of message M*
M.nsrc = *network layer source endpoint of message M*
M.ndst = *network layer destination endpoint of message M*
M.reqid = *request identifier of message M*
M.time = *capture timestamp of message M*
M.oids = *OIDs listed in varbind list of message M*
M.values = *values listed in varbind list of message M*

These properties of an SNMP message can be easily extracted from the exchange formats defined in [3].

Definition 1. *This definition establishes the following message classes:*

1. *A **read request message** is a message M containing a PDU of type GetRequest, GetNextRequest, or GetBulkRequest.*
2. *A **write request message** is a message M containing a PDU of type SetRequest.*
3. *A **notification request message** is a message M containing a PDU of type InformRequest.*
4. *A **notification message is a message** M containing a PDU of type Trap or InformRequest.*
5. *A **request message is a message** M which is either a read request message, a write request message, or a notification request message.*
6. *A **response message** is a message M containing a PDU of type Response or of type Report.*
7. *A **non-response message** is a message M which is either a read request message, a write request message, or a notification message.*
8. *A **command message** is a message M which is either a read request message or a write request message.*

Report messages are treated like Response messages since the SNMPv3 specifications currently use Report messages only as an error reporting mechanism, always

triggered by the processing of some request messages. In case future SNMP versions or extensions use Report messages without having a request triggering the generation of Report messages, we may have to revisit the definition above.

Definition 2. *A set of* ***command group messages*** *consists of all messages M satisfying either of the following two conditions:*

1. *M is a command message*
2. *M is a response message and there exists a command message C such that the following holds:*

$$\begin{array}{ll} M.reqid & = C.reqid \\ M.tdst & = C.tsrc \\ M.tsrc & = C.tdst \\ (M.time - C.time) & < t \end{array}$$

The parameter t defines a maximum timeout for response messages.

This definition requires that the response message originates from the transport endpoint over which the request message has been received. This is not strictly required by SNMP transport mappings and in particular the UDP transport mapping allows to send responses from different transport endpoints. While sending response messages from a different transport endpoint is legal, it is also considered bad practice causing interoperability problems, since some management systems do not accept such messages.

It was decided to require matching transport endpoints since doing so significantly simplifies the procedures below and avoids accidentally confusing requests and responses. Implementations responding from different transport endpoints will lead to (a) a larger number of requests without related responses (and likely no retries) and (b) a similarly large number of responses without a matching request. If such behavior can be detected, the traces should be investigated and if needed the transport endpoints corrected. The requirement for matching transport endpoints only affects request / response pairs. It is perfectly fine for a manager to use different transport layer endpoints in different polling instances, or even different operations (i.e., slices) within the same polling instance.

Definition 3. *A set of* ***notification group messages*** *consists of all messages M satisfying either of the following two conditions:*

1. *M is a notification message*
2. *M is a response message and there exists a notification request message N such that the following holds:*

$$
\begin{array}{ll}
M.reqid & = N.reqid \\
M.tdst & = N.tsrc \\
M.tsrc & = N.tdst \\
(M.time - N.time) & < t
\end{array}
$$

The parameter t defines a maximum timeout for response messages.

This definition again requires matching transport endpoints for notification

5.5 Traces and Flows

Traces are (large) sets of SNMP messages that are the result of recording SNMP traffic using a single traffic recording unit (e.g., using tcpdump) on a network segment carrying traffic of one or more managers and agents. Traces being used in the remainder of this document may be altered as a result of anonymization, which may result in some message information loss.

Traces may contain SNMP messages that have been exchanged between possibly many different network layer endpoints. One way of making an initial separation of such a trace into more manageable pieces is by splitting the messages into flows. Each flow contains only messages that have occurred between two network layer endpoints.

5.5.1 Trace and Flow Definition

Definition 4. *An SNMP **trace** (or short trace) T is an ordered set of zero or more SNMP messages M. All messages M in T are chronologically ordered according to the capture timestamp M.time.*

Definition 5. *A **flow** F is the set of messages of an SNMP trace T with the following properties:*

1. *All response messages originate from a single network endpoint.*
2. *All non-response messages originate from a single network endpoint.*
3. *All messages are either command group messages with parameter t or notification group messages with parameter t.*

Parameter t defines the maximum timeout for response messages. The value of t should be chosen such that only response messages to the respective nonresponse messages are considered part of the same flow. Analysis of a large number of traces shows that 25 seconds is a proper default value for t.

It is possible that response messages of a trace cannot be classified to belong to any flow. This can happen if request messages triggering the response messages were not recorded (for example due to asymmetric routing), or because response messages were originating from transport endpoints different from the endpoint used to receive the associated request message.

This definition of a flow indicates that it can be either unidirectional (e.g., a manager sending non-response messages to a non-responding agent), or bidirectional (e.g., a manager reading a table from an agent). This is different from other flow definitions, like the NetFlow definition [5]. The flow definition is mostly consistent with the definition of an SNMP flow used in [6]. The difference is that the tool used to generate the data reported in [6] did only require that the network layer source endpoint of the response messages matches the destination network layer endpoint of the associated request messages.

Definition 6. *A **flow initiator** is the network layer endpoint of the two endpoints involved in a flow, which is responsible for sending the first non-response message.*

Notation 2. *The properties of a flow F are denoted as follows:*

$F.type$ = *type of the flow F (command/notification)*
$F.nsrc$ = *network layer source endpoint of F*
$F.ndst$ = *network layer destination endpoint of F*
$F.start$ = *timestamp of the first message in F*
$F.end$ = *timestamp of the last message in F*
$F.init$ = *initiator of the flow F*
$F.t$ = *parameter t of F (maximum timeout for response messages)*

Subsequently, flows containing only command group messages are called command flows. Similarly, flows containing only notification group messages are called notification flows.

5.5.2 Trace and Flow Example

Table 5.1 shows an example of a trace consisting of SNMP messages that were exchanged between different network layer endpoints. The network layer endpoints are represented by A, B, C and D.

The first flow F_1 consists of SNMP messages that have been exchanged between network layer endpoints A and B, where all response messages originate from B and all non-response messages originate from A. The minimum value of parameter t for this flow is 0.07 seconds, since that is the longest time between a request and its subsequent response message.

Message	Time [s]	Direction	Type	Reqid	Flow
0	0.00	A → B	GetNext	1	F_1
1	0.04	C → D	Get	10	F_2
2	0.05	B → A	Response	1	F_1
3	0.08	D → C	Response	10	F_2
4	0.11	A → B	GetNext	2	F_1
5	0.15	B → A	Response	2	F_1
6	0.18	A → B	GetNext	3	F_1
7	0.22	D → C	Trap	14	F_3
8	0.25	B → A	Response	3	F_2

TABLE 5.1: Example trace containing two flows

The second flow F_2 contains SNMP messages exchanged between network layer endpoints C and D, where all response messages originate from D and all non-response messages originate from C. The minimum value of parameter t for this flow is 0.04 seconds.

The third flow F_3 contains the remaining SNMP messages of the trace that occurred between network layer endpoints C and D. In this case the non-response message originates from D. There is no parameter t applicable to this flow, because there are no response messages.

5.6 Slices

Flows can still contain a large amount of SNMP messages. A flow should therefore be split up into even smaller sets of messages. One way of identifying meaningful subsets of messages of a flow would be by considering the behavior of managers and agents. In the case of managers, they are usually configured to perform regular polling instances. In such a polling instance, the manager might poll a number of agents. Since a flow contains only the messages exchanged between two network layer endpoints, a flow therefore probably consists of only a subset of the messages that are part of a polling instance. So, one option of finding smaller, meaningful subsets of messages within flows, would be by looking for messages that belong to a particular polling instance. Such a smaller set of messages is called a slice.

5.6.1 Slice Definition

Definition 5. *A **flow** F is the set of messages of an SNMP trace T with the following properties:*

1. All response messages originate from a single network endpoint.

2. *All non-response messages originate from a single network endpoint.*
3. *All messages are either command group messages with parameter t or notification group messages with parameter t.*

Definition 7. *A **slice** S with parameter e is a subset of messages in a flow F for which the following properties hold:*

1. *All messages are exchanged between the same two transport endpoints (a single transport endpoint pair).*
2. *All non-response messages must have a PDU of the same type.*
3. *All messages with a PDU of type Get, Set, Trap, or Inform must contain the same set of OIDs.*
4. *Each GetNext or GetBulk message must either contain the same set of OIDs as the preceding request or it must be linked to a response of the last previously answered request (i.e., the request must contain at least one OID that has been contained in the (repeater) varbind list of a preceding response message of the last answered request message).*
5. *All Response messages must follow a previous request message that is part of the same slice.*
6. *For any two subsequent non-response messages Q1 and Q2 with Q1.time < Q2.time, the following condition must hold:*

$$(Q2.time - Q1.time) < e$$

The first item in the slice definition requires that the messages of a single slice are exchanged between a single transport layer endpoint pair. This is different from the flow definition, which requires a single network layer endpoint pair. The choice of looking at the transport layer endpoints in the case of slices is based on the assumption that, for instance, multiple managers and agents might be operating from the same respective network layer endpoint. Another assumption is that a manager and an agent will only use a single transport layer endpoint respectively when they communicate for the duration of a slice (or even a polling instance). A previous section already mentioned some issues when a manager or agent uses different transport layer endpoints within a single polling instance.

The parameter e defines the maximum time between two non-response messages that belong to a slice. This parameter should be chosen such that unrelated non-response messages within a flow are not considered to be of the same slice. Unrelated non-response messages are those that, for instance, belong to different polling instances. The parameter e should therefore be larger than the retransmission interval in order to keep retransmissions within a slice and smaller than the polling interval used by the slice initiator.

The value of parameter e might be closely related to parameter t of the respective flow the slice is part of. For instance, if parameter e is very large, than t is also likely to be very large and vice versa. Also, if parameter e is very small, then t is probably also very small. However, it is not possible to strictly state that e and t are always closely related to each other, because parameter e is specific for a slice. This is in contrast with parameter t which is specific for a much larger set of messages, a flow.

Definition 8. *A **slice initiator** is one of the two transport layer endpoints involved in a slice, which is responsible for sending the chronologically first nonresponse message.*

Notation 3. *The properties of a slice S are denoted as follows:*

$S.type$ = *type of non-response messages in S*
$S.tsrc$ = *transport layer endpoint of initiator of S*
$S.tdst$ = *transport layer endpoint of non-initiator of S*
$S.start$ = *timestamp of the chronologically first message in S*
$S.end$ = *timestamp of the chronologically last message in S*
$S.init$ = *initiator of slice S*
$S.e$ = *parameter e of S (maximum time between two non-response messages)*

5.6.2 Slice Example

Table 5.2 shows an example of a flow containing messages exchanged between transport layer endpoints A, B, and C. Considering the timing of the messages, a proper value of e should be $0.14 \leq e \leq 299.82$ seconds. Such a value for parameter e will separate the flow into two apparent polling instances, which each contain the same set of messages.

The first slice S_1 consists of a Get request and its subsequent response. A similar request is recorded later in slice S_3 but since we assume e as discussed above, the slices S_1 and S_3 are distinct. The second slice S_2 also contains a Get request and its subsequent response. This slice is different from S_1 since a different OID is requested. The slice S_4 consists of a Set request that has not been answered. (A potential reason is that the SNMP engine listening on the transport layer endpoint C did not grant write access and dropped the message.)

The last slice S_5 contains a sequence of linked GetNext requests. The GetNext request message 10 is likely a retransmission of the GetNext request message 9. This example demonstrates that retransmissions are recorded in the slice that contains the original request.

Message	Time [s]	Direction	Type	Reqid	OIDs	Slice
0	0.00	A → B	Get	1	alpha.1	S_1
1	0.06	B → A	Response	1	alpha.1	S_1
2	0.12	A → B	Get	1	beta.1	S_2
3	0.17	B → A	Response	1	beta.1	S_2
4	300.00	A → B	Get	1	alpha.1	S_3
5	300.05	B → C	Set	1	gamma.1	S_4
6	300.07	B → A	Response	1	alpha.1	S_3
7	300.14	A → B	GetNext	1	beta	S_5
8	300.19	B → A	Response	1	beta.1	S_5
9	300.32	A → B	GetNext	1	beta.1	S_5
10	300.52	A → B	GetNext	1	beta.1	S_5
11	300.58	B → A	Response	1	delta.1	S_5

TABLE 5.2: Example flow containing three slices

5.7 Slice Signatures and Prefix

As noted in the beginning of this document, it is desirable that slices can be tested for equality/comparability. This is where the slice prefix comes in. The slice prefix provides one of the means to compare slices. Using the slice prefix and a few other parameters of a number of slices, one can determine which slices should be considered equal and which of them are incomparable. This will assist in the process of finding potentially other relations.

The slice prefix is a set of OIDs. This set is constructed from the messages that make up a single slice. So, for example, a slice that is the result of a manager requesting the contents of a particular table (with OID alpha) on an agent using a simple single varbind GetNext walk, starting at the table OID alpha, shall yield a slice prefix which consists of the OID alpha.

Because the aim is to compare various slices using the slice prefix (along some other characteristics of a slice), this implicitly suggests the need to know whether a number of slices are the result of the same behaviour (i.e., specific configuration) of the initiating party of these slices. For example, one may want to know whether a number of slices that involve a single manager and a single agent were the result of just one specific configuration of that manager. Multiple slices, that may all be initiated by that same manager and each slice possibly occurred in different polling instances, may in fact be the result of the same specific configuration of that particular manager. So, since in this case the specific configuration of the manager is only relevant for determining the behaviour, the slice prefix should be constructed based on OIDs in messages originating from that manager only. More generally, only the messages within slices that are sent by the initiating party (the non-response messages) are considered for the determination of the respective slice prefix of a slice.

5.7.1 Slice Signatures and Prefix Definition

Definition 9. *A **slice signature** S.sig of a slice S is a set of OIDs derived from the OIDs contained in the non-response messages of a slice. Let r(S) denote the set of response messages of slice S and n(S) the set of non-response messages of S. Then the set S.sig consists of the following OIDs:*

$$S.sig = \begin{cases} \bigcup_{n \in n(S)}(n.oids) \setminus \bigcup_{r \in r(S)}(r.oid) & S.type \text{ is GetNext or GetBulk} \\ \bigcup_{n \in n(S)}(n.oids) & otherwise \end{cases}$$

The slice signature summarizes which OIDs have been carried in a slice and is straightforward to compute. However, there are situations in some GetNext or GetBulk sequences where the signature might contain some unwanted OIDs as will be demonstrated by an example below.

To further condense signatures, it is necessary to introduce a prefix relationship between OIDs. This prefix relationship can then be used to reduce a slice signature to a slice prefix.

Definition 10. *An OID a = a1.a2...an is a **prefix** of OID b = b1.b2...bm if and only if $n < m$ and $a_i = b_i$ for $1 \leq i \leq n$.*

Definition 11. *The **slice prefix** S.slice is the set of all OIDs o in S.sig for which there is no p in S.sig such that p is a prefix of o.*

5.7.2 Slice Signatures and Prefix Example

The following example demonstrates how a slice prefix is determined. Consider the case that a single manager A is set to poll a specific agent B. Manager A is programmed to retrieve some values from B. A single slice may contain the messages shown in Table 5.3.

The slice shown in Table 5.3 has a number of interesting properties. First, not all columns in the retrieved table have an equal length. Second, the manager is set to request the sysUpTime on an irregular basis (i.e., every few requests). Third, the manager attempts to fill holes in the table and finally the order of referenced OIDs in GetNext messages changes.

All of these properties do not influence the process for determining the slice signature. The slice prefix is constructed as follows:

Message	Direction	Type	OIDs
0	A → B	GetNext	alpha, beta
1	B → A	Response	alpha.1, beta.1
2	A → B	GetNext	alpha.1, beta.1
3	B → A	Response	alpha.2, beta.3
4	A → B	GetNext	beta.2, alpha.2, sysUpTime
5	B → A	Response	beta.3, alpha.3, sysUpTime.0
6	A → B	GetNext	beta.3, alpha.3
7	B → A	Response	gamma.1, alpha.4
8	A → B	GetNext	alpha.4
9	B → A	Response	delta.1

TABLE 5.3: Example slice for calculating a slice prefix

1. The union of all OIDs in non-response messages is the following set:

   ```
   N = { alpha, beta, alpha.1, beta.1, beta.2, alpha.2,
         sysUpTime, beta.3, alpha.3, alpha.4 }
   ```

2. The union of the OIDs in response messages is the following set:

   ```
   R = { alpha.1, beta.1, alpha.2, beta.3, alpha.3,
         sysUpTime.0, gamma.1, alpha.4, delta.1 }
   ```

3. Subtracting the two sets results in the slice signature S.sig:

   ```
   S.sig = N - R = { alpha, beta, beta.2, sysUpTime }
   ```

   The element beta.2 exists, because the manager was trying to fill a "hole" in a table. Since these holes reside in the tables on the agent side and may change dynamically, they do not really help in describing the behavior of the initiating party.

4. Since beta is a prefix of beta.2, the slice prefix becomes the following set:

   ```
   S.prefix = { alpha, beta, sysUpTime }
   ```

   The slice prefix does not include beta.2 anymore and thus a manager retrieving the same columns alpha and beta with and without holes will produce slices with the same slice prefix.

5.8 Slice Types

As described previously, the slice type allows for comparing slices. This means that any number of slices that are of the same slice type may be considered an equivalence class and may therefore be considered to be the result of the same behaviour of the slice initiator.

5.8.1 Slice Type Definition

Definition 12. *Two slices A and B satisfy the binary* ***slice equivalence*** *relation* $A \sim B$ *if the following properties hold:*

1. *All messages in A and B have been exchanged between the same network layer endpoints.*
2. *All read request messages, write request messages, and notification messages in A and B originate from the same network layer endpoint.*
3. *All non-response messages in A and B are of the same type.*
4. *The slices A and B have the same prefix, that is A.prefix = B.prefix.*

It can be easily seen that the relation $\sim$ is reflexive, symmetric, and transitive and thus forms an equivalence relation between slices.

Definition 13. *Let S be a set of slices, then all slices in the equivalence class*

$$[\mathrm{A}] = \{\mathrm{s} \in \mathrm{S} \mid \mathrm{s} \sim \mathrm{A}\}$$

with $A \in S$*, are of the same* ***slice type****.*

5.8.2 Slice Type Example

The flow shown in Table 5.4 contains two slices. The first slice S_1 contains messages that have been exchanged between transport layer endpoints A and B while the second slice S_2 contains messages that have been exchanged between transport layer endpoints C and D. However, the network layer endpoints of this slice are the same as the first slice and all non-response messages in both slices originate from the same network layer endpoint.

As can be verified easily, the slices S_1 and S_2 satisfy the slice equivalence relationship, that is $S_1 \sim S_2$ and they form an equivalence class under $\sim$, which we call a slice type.

Message	Direction	Type	OIDs	Slice
0	A → B	GetNext	alpha, beta	S_1
1	B → A	Response	alpha.1, beta.1	S_1
2	A → B	GetNext	alpha.1, beta.1	S_1
3	B → A	Response	alpha.2, beta.3	S_1
4	A → B	GetNext	beta.2, alpha.2, sysUpTime	S_1
5	B → A	Response	beta.3, alpha.3, sysUpTime.0	S_1
6	A → B	GetNext	beta.3, alpha.3	S_2
7	B → A	Response	gamma.1, alpha.4	S_2
8	A → B	GetNext	alpha.4	S_2
9	B → A	Response	delta.1	S_2

TABLE 5.4: Example for slice equivalence and slice types

5.9 Related and Future Work

The performance of SNMP has been the subject of several studies. Some papers such as [7, 8] compare the performance of centralized SNMP management to distributed management approaches while other papers compare the performance of the SNMP protocol with middleware systems such as CORBA or Web Services [911]. The impact of security protocols on SNMP performance has been studied in [1214]. Some authors have formulated models for the SNMP protocol [15, 16].

All these studies have in common that they make assumptions how SNMP is used in practice, due to a lack of commonly accepted models how SNMP is used in practice. To address this issue, some researchers active in the Network Management Research Group (NMRG) of the Internet Research Task Force (IRTF) started an effort to collect traces from operational networks and to build the necessary tools to analyze them [3]. First results were published in [6] and it became clear that precise definitions of basic concepts, such as those provided in this paper, are needed in order to produce meaningful and comparable results. The authors are currently using and extending these definitions in order to study the periodicity of SNMP traffic [17] and table retrieval algorithms.

5.10 Conclusions

The analysis of SNMP traces requires a collection of common and precise definitions in order to establish a basis for producing meaningful and comparable results. This paper provides such a collection of basic definitions that have been developed over time and are also being reviewed and discussed within the NMRG of the IRTF. This paper is a condensed summary of a more detailed document [18] submitted to the NMRG and written to provide an early stable reference while the

work in the NMRG continues and to foster discussion within the broader network management research community.

5.11 Acknowledgement

The work reported in this paper is supported by the EC IST-EMANICS Network of Excellence (#26854).

5.12 References

1. Case, J., Mundy, R., Partain, D., Stewart, B.: Introduction and Applicability Statements for Internet Standard Management Framework. RFC 3410, SNMP Research, Network Associates Laboratories, Ericsson (December 2002)
2. Harrington, D., Presuhn, R., Wijnen, B.: An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. RFC 3411, Enterasys Networks, BMC Software, Lucent Technologies (December 2002)
3. Schonwalder, J.: SNMP Traffic Measurements and Trace Exchange Formats. Internet Draft (work in progress) ¡draft-irtf-nmrg-snmp-measure-04.txt¿, Jacobs University Bremen (March 2008)
4. Presuhn, R.: Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP). RFC 3416, BMC Software (December 2002)
5. Claise, B.: Cisco Systems NetFlow Services Export Version 9. RFC 3954, Cisco Systems (October 2004)
6. Schonwalder, J., Pras, A., Harvan, M., Schippers, J., van de Meent, R.: SNMP Traffic Analysis: Approaches, Tools, and First Results. In: Proc. 10th IFIP/IEEE International Symposium on Integrated Network Management. (May 2007)
7. Zapf, M., Herrmann, K., Geihs, K.: Decentralized SNMP Management with Mobile Agents. In: Proc. 6th IFIP/IEEE International Symposium on Integrated Network Management, Boston (May 1999) 623 – 635
8. Fuggetta, A., Picco, G., Vigna, G.: Understanding Code Mobility. IEEE Transactions on Software Engineering 24(5) (May 1998) 342 – 361
9. Gu, Q., Marshall, A.: Network Management Performance Analysis and calability Tests: SNMP vs. CORBA. In: Proc. 2004 IEEE/IFIP Network Operations and Management Symposium, Seoul (April 2004) 701 – 714
10. Pras, A., Drevers, T., van de Meent, R., Quartel, D.: Comparing the Performance of SNMP and Web Services based Management. IEEE Transactions on Network and Service Management 1(2) (November 2004)
11. Pavlou, G., Flegkas, P., Gouveris, S., Liotta, A.: On Management Technologies and the Potential of Web Services. IEEE Communications

Magazine 42(7) (July 2004) 58 – 66
12. Du, X., Shayman, M., Rozenblit, M.: Implementation and Performance Analysis of SNMP on a TLS/TCP Base. In: Proc. 7th IFIP/IEEE International Symposium on Integrated Network Management, Seattle (May 2001) 453 – 466
13. Corrente, A., Tura, L.: Security Performance Analysis of SNMPv3 with Respect to SNMPv2c. In: Proc. 2004 IEEE/IFIP Network Operations and Management Symposium, Seoul (April 2004) 729 – 742
14. Marinov, V., Schonwalder, J.: Performance Analysis of SNMP over SSH. In: Proc. 17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2006). Number 4269 in LNCS, Dublin, Springer (October 2006) 25 – 36
15. Pattinson, C.: A study of the behaviour of the simple network management protocol. In: Proc. 12th IFIP/IEEE Workshop on Distributed Systems: Operations and Management, Nancy (October 2001)
16. Chen, T., Liu, S.: A Model and Evaluation of Distributed Network Management Approaches. IEEE Journal on Selected Areas in Communications 20(4) (May2002) 850 – 857
17. van den Broek, J.G.: Periodicity of SNMP Traffic. BSc Thesis (August 2007)
18. van den Broek, J.G., Schonwalder, J., Pras, A., Harvan, M.: SNMP Trace Analysis Definitions. Internet Draft (work in progress) <draft-schoenw-nmrg-snmp-tracedefinitions-02.txt>, University of Twente, Jacobs University Bremen, ETH Zurich (April 2008)

Chapter 6

Paper - Periodicity of SNMP Traffic

Authors: J.G. van den Broek (University of Twente), J. Schönwälder (Jacobs University Bremen), A. Pras (University of Twente)

This paper is to be published. This paper has been edited for style.

6.1 Abstract

The Simple Network Management Protocol (SNMP) is currently widely deployed to monitor, control, and configure network elements. The SNMP traffic that takes place between these network elements can be characterized as periodic or aperiodic. Periodic traffic could be the result of a programmed manager, like MRTG, to perform regular polling actions. Aperiodic traffic could be manually induced by a network operator. It is assumed that periodic traffic shows different characteristics than aperiodic traffic. To identify these characteristics, we first need to be able to separate these two kinds of traffic from each other. This paper discusses a possible methods through which these two kinds of SNMP traffic can be separated. It proposes an algorithm that is subsequently implemented in a toolset. This toolset can in turn be used to split the SNMP traffic in periodic and aperiodic traffic. The analysis of this toolset suggests that the developed algorithm is capable of separating the two traffic types in nearly all cases.

6.2 Introduction

The Simple Network Management Protocol (SNMP) was introduced in the late 1980s. Since then, a lot of protocol changes have taken place, which have eventually resulted in what is known today as the SNMP version 3 framework (SNMPv3) [1, 2]. Since then, extensive use of SNMP has led to significant practical experience within this field by both network operators and researchers. But until now, only little analysis has been done on SNMP traffic due to a lack of real world SNMP traces. Traces are network traffic recordings using tools such as *tcpdump*.

These days, some researchers managed to collect a wide variety of SNMP traces. These traces contain traffic that is the result of either periodic, or aperiodic behavior. Periodic traffic that is, for instance, generated by a manager polling a set of agents every few minutes, might have interesting characteristics that differ from aperiodic traffic. Both kinds of traffic presumably have different characteristics. Knowing about them might assist network operators in identifying previously unknown problems. Moreover, specific research on each of these kinds of traffic might be performed. However, no proper method for doing so within the context of SNMP is known today.

Therefore, a number of research questions need to answered. The following questions are subsequently answered in this paper:

- **Which approaches to separate periodic from aperiodic SNMP traffic are possible?**
- **Which SNMP properties characterize a working solution?**
- **How can periodicity be determined?**
- **What periodicity can be found in lab and real world traces?**

The first question to be answered in this paper is about which approaches might be feasible. After considering possible approaches, one approach is chosen in chapter 2. An introductory description of the chosen approach is given in chapter 3 by describing it in general, without focusing on SNMP. This also suggests the general applicability of this approach. The chosen approach is then described specifically for SNMP in terms of how periodic and aperiodic traffic can be identified and subsequently separated in chapters 4 and 5. Chapter 4 focuses on finding relations in SNMP traffic and chapter 5 attempts to find periodicity between these relations. After that, we will verify the approach by applying the developed toolset on artificial and lab traces that are created in a known environment. The results of this verification process are presented in chapter 6. Then we will apply the tools on real world traces in chapter 7 and end with concluding remarks and future work suggestions in chapter 8.

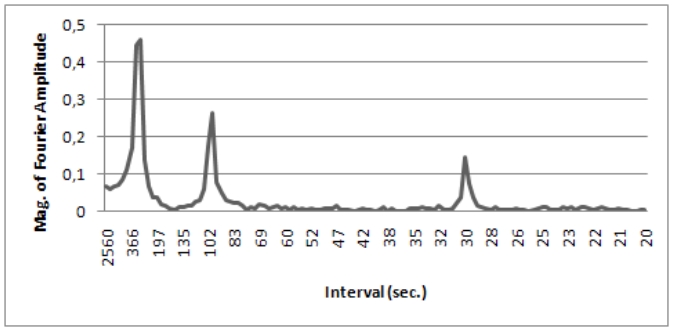


FIGURE 6.1: Fourier analysis of small artificial SNMP trace

6.3 Possible approaches

Distinguishing between periodic and aperiodic traffic might be possible using a number of approaches. Following is a short description of three potential approaches, including the one described in the remainder of this paper.

6.3.1 Fourier Analysis

Until now, hardly any research has been carried out to determine periodicity of SNMP traffic. Papers like [3] describe tools to analyze SNMP traces, but mention only briefly the occurrence of specific (periodic) patterns.

Extensive use of Fourier analysis can be found in the field of network security, as elaborated upon in [4, 5]. These papers describe the use of Fourier analysis on network traffic with the goal to detect specific network threats. Fourier analysis is also used in detecting network traffic anomalies [7]. Besides that, Fourier analysis has also been used to identify periodicity in internet traffic in general [10]. Within the field of SNMP related research, one can find [8]. That paper describes some analysis of SNMP traffic and describes an attempt to make use of Fourier analysis in order to find periodicity within SNMP traces by detecting peaks.

Using Fourier analysis to separate periodic from aperiodic SNMP traffic might thus be one of the possible approaches. In our research the applicability of this approach is tested using an artificial SNMP trace. Fig. 6.1 shows the magnitude of Fourier amplitude versus the interval time of SNMP message occurrences within such an SNMP trace. Here we can see distinct peaks around 300, 100 and 30 seconds. The peak around 300 seconds is probably the result from the fact that most SNMP manager applications are set to poll agents using a 300 second interval. But finding the exact cause of any peak would require identifying the responsible SNMP messages. However, it is not possible to determine the responsible messages for a specific peak exactly with this approach.

6.3.2 Reverse engineering

Another potential approach would involve reverse engineering [6] of some well known SNMP manager and agent implementations. Although some are open source, like *Cacti* and *MRTG*, which would make it easy to understand the exact behavior of these tools, others are not, like *HP's Open View*. Many other closed source implementations exist and are in use today. Therefore, reverse engineering these closed source tools and analyzing the source of the open source tools could give us information about specific traffic patterns we might find in traces.

However, reverse engineering would only be a first step, since the resulting patterns subsequently need to be detected in traces using some toolset.

Besides the fact that reverse engineering is a cumbersome task, it is also practically impossible to include characteristics of all known SNMP implementations, let alone keeping these characteristics up-to-date over time. A more generic approach would therefore be preferable.

6.3.3 Finding patterns in sets of messages

The third possibility is a generic approach, which encompasses finding relations between sets of messages firs and subsequently attempting to find periodic patterns between these sets.

The difference between this approach and reverse engineering would be the lack of actually performing reverse engineering, by merely looking for relations between messages in a generic way. This would be done by establishing and finding fundamental relations between SNMP messages, like those defined in [11]. However, those defined relations only form a small foundation for finding even larger relations that need to be found in order to say anything conclusive about whether or not messages are to be considered periodic or aperiodic.

This last approach is the chosen one, because it works on a per message basis, which contrasts the application of Fourier analysis. Also, this approach would be more scalable than reverse engineering, since it would not involve specific characteristics of any SNMP implementation. On the other hand, finding potentially complex relations might result in a more complex toolset than might be case for reverse engineering. The following chapters describe this approach by starting with a generic description of the used terminology, which in turn shows that this approach is broadly applicable.

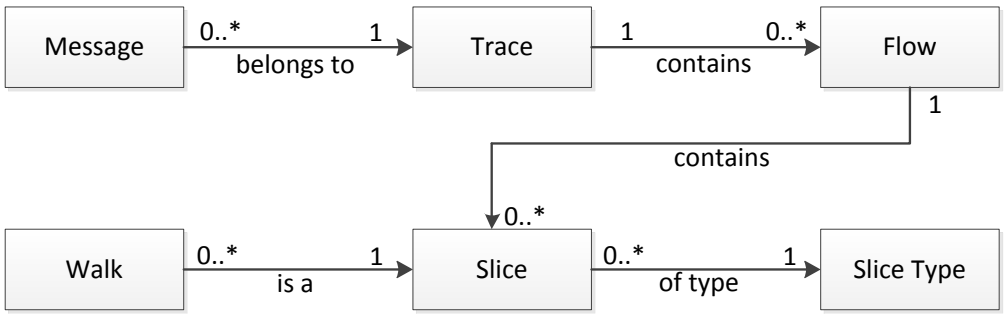


FIGURE 6.2: Relationship between messages

6.4 Relations in network traffic

Our approach involves finding relations between exchanged SNMP messages that can be found in a trace. Fig. 6.2 shows a number of possible identifiable relations for SNMP traffic. Some of these relations are extensively described in paper, others however are new. For explanatory purposes, this chapter shows that these relations can be generalized and applied to completely different protocols. An intuitive description of each of these relations is therefore given in the context of two protocols: *HTTP* and *DNS*. The choice for HTTP is based on its simple nature and the fact that it is well known. DNS is a bit more elaborate, which allows us to work towards the characteristics of SNMP in the next chapter.

6.4.1 Messages and Traces

The largest relation possible is depicted in Fig. 6.2 as the trace. A trace consists of zero or more messages. A trace should therefore be seen as a large set of messages that is the result of a recording of network traffic on a specific network segment. Although [11] describes the trace in the context of SNMP messages, it is also possible to apply this concept to HTTP traffic, where only HTTP traffic is recorded into a single trace. We can argue similarly for DNS traffic.

On the other hand, the smallest unit is a single message. The header and possibly the payload combined of each message can later on be used to find inter-message relations.

6.4.1.1 HTTP

In the case of HTTP traffic a single message could be a HTTP request message sent by a client to a server, as illustrated in the trace shown in Fig. 6.3. The

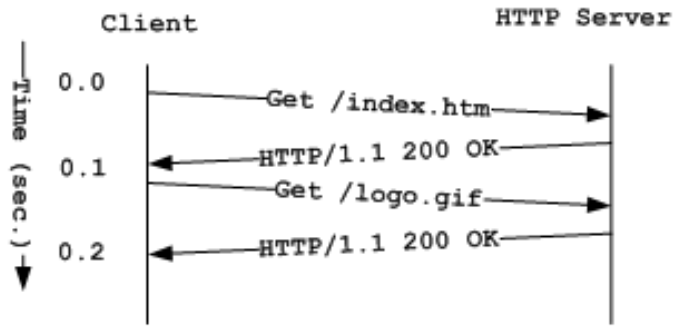


FIGURE 6.3: Example exchange of HTTP messages (1)

simple case where a web browser (client) requests a webpage and a single image related to that webpage is shown. After setting up a TCP connection to the server, the client first requests index.htm, which is followed by a specific response from the server. This response is the contents of the requested file. After obtaining `index.htm`, the client requests `logo.gif`, which is subsequently sent by the server to the client. The four messages shown, each depicted as a single arrow, make up the complete exchange of messages between the client and the web server that are necessary for the client to obtain the two files.

6.4.1.2 DNS

A similar description of messages in a trace can be made for the DNS protocol. Fig. 6.4 shows a simplified scenario where a requesting host wants to have the IP address of a resource identified with $\alpha.\beta.\gamma$. The first message in this example is a query message sent to a local DNS server with identification value 0. The local DNS server in turn attempts to resolve a part of the referenced resource name by sending a query to a TLD DNS server it knows is relevant for resolving part of the name. This second query has identifier 1. The TLD DNS server is apparently able to resolve this portion of the resource name and replies with a query response, having the same identifier as the query sent to it by the local DNS server. Subsequently, the local DNS server sends a query to the relevant authoritative DNS server in order to resolve the resource name completely. Again here a single identifier (2) is used in the query and query response. After receiving the query response from the authoritative DNS server, the local DNS server returns the result to the requesting host using the same identifier used in the initial query sent by the requesting host. As a result, each of the shown arrows represents a single message in a larger trace consisting of only DNS traffic.

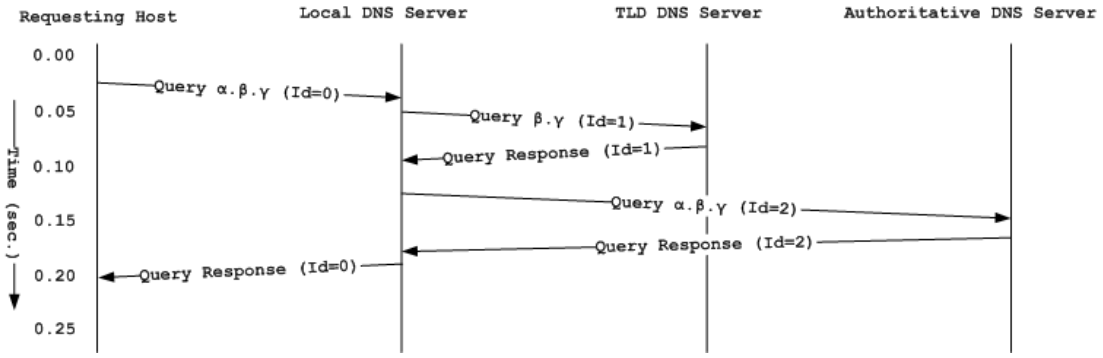


FIGURE 6.4: Example exchange of DNS messages (1)

6.4.2 Flows

Traces are merely a collection of individual messages. In order to find other relations between messages within a trace, it would be desirable to separate a trace into a number of sets of messages based on a very rough separation criterion. Fig. 6.2 shows that a flow would be such a relation. A flow in the context of SNMP consists of messages that have been exchanged between the same two network layer endpoints (i.e., have the same IP address) [11] and may contain messages going in both directions. This definition is clearly different from the definition of *NetFlow*, which consist only of messages going in one direction. The concept of flows can also be generalized and be applied to the two other protocols.

6.4.2.1 HTTP

The scenario shown in Fig. 6.3 is already a single flow: all four messages are exchanged between the same two entities, namely the client (network layer endpoint α) and the web server (network layer endpoint β). In this case, the single flow encompasses all messages of a single trace. So, all HTTP messages occurring between a single client and a single server could be seen as an HTTP flow.

6.4.2.2 DNS

The flows that can be found in the messages in the trace shown in Fig. 6.4 are between the requesting host and the local DNS server, or the local DNS server and the other DNS servers respectively. A total of three flows could be identified in this trace, each containing two messages. A typical DNS flow would therefore consist of all DNS messages occurring between two DNS nodes, like a workstation and a DNS server, or two DNS servers.

6.4.3 Slice

Within a flow, one might be able to identify smaller sets of messages that appear to be related. In [11], where slices make up a flow, other protocols also show similar smaller sets of messages within flows. Although a slice in SNMP contains, for instance, a get-request and a related response, a generalized description of a slice would involve any number of messages within a flow that are related to a particular action (e.g., a request and the related response).

6.4.3.1 HTTP

The first message shown in Fig. 6.3 is a request message sent by the client. The web server in turn responds to that particular request. The third message is again a request message sent to the web server, which again responds with the contents of the relevant file. This suggests that two request/response-pairs might be identified in this case. The first such pair consists of the first two messages and the second pair of the last two messages. In turn two slices might be identified in this single flow.

6.4.3.2 DNS

Between the messages shown in Fig. 6.4 that occur between the same two network layer endpoints (i.e., within a single flow), it is also possible to relate a query and a response to each other. For instance, of the messages that occurred between the requesting host and the local DNS server, it is possible to relate the query with identifier 0 with the query response with identifier 0. Similar relations can be made between the local DNS server and the TLD and Authoritative DNS server respectively. Note that one should not consider queries and query responses with the same identifier to be related when they occurred in different flows, because identifiers in DNS allow querying hosts to match queries with query responses (i.e., an identifier in DNS need not be unique in the whole network, let alone a trace). This example also stresses the need to enforce constraints on the maximum amount of time between a query and a response before they are considered related in the form of a slice. Such a constraint is also applied in the case of an SNMP slice [11].

6.4.4 Slice Type

Each slice can be the result of an action that may occur more than once. It is therefore likely that a trace might contain slices that are the result of the same

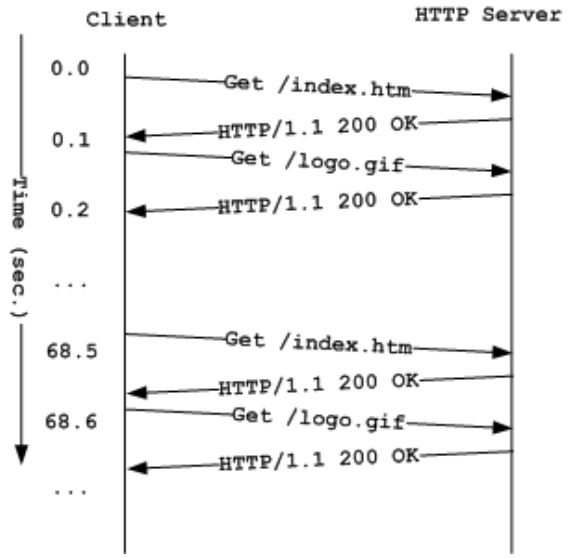


FIGURE 6.5: Example exchange of HTTP messages (2)

behavior of the initiating party and may therefore be comparable. However, determining whether any two slices are to be considered comparable, and hence of the same slice type, may differ significantly between the various example protocols.

6.4.4.1 HTTP

The described HTTP scenario can be extended by considering the option that the client might send a request for the same webpage and image at a later point in time. This could occur, because the page is requested again by the client user and the current page and image in the cache at the client have expired. This would result in a similar set of messages that would take place between the client and the web server. Fig. 6.5 shows such a scenario.

The second set of messages (the third and fourth slices shown) - that occurs around 68 seconds into the trace and the same flow - are highly comparable to the first respective ones, since it involves the exact same webpage and image. The third slice consists of the second request to obtain the webpage and image (fourth slice) are the same in quantity of messages, but that does not have to be the case in general. Besides the quantity of messages, many other differences might exist. Moreover, the requested files in the third slice are the same by name as in the first slice. A simple way of comparing various HTTP slices occurring between the client and the web server could therefore involve comparing the requested files by name.

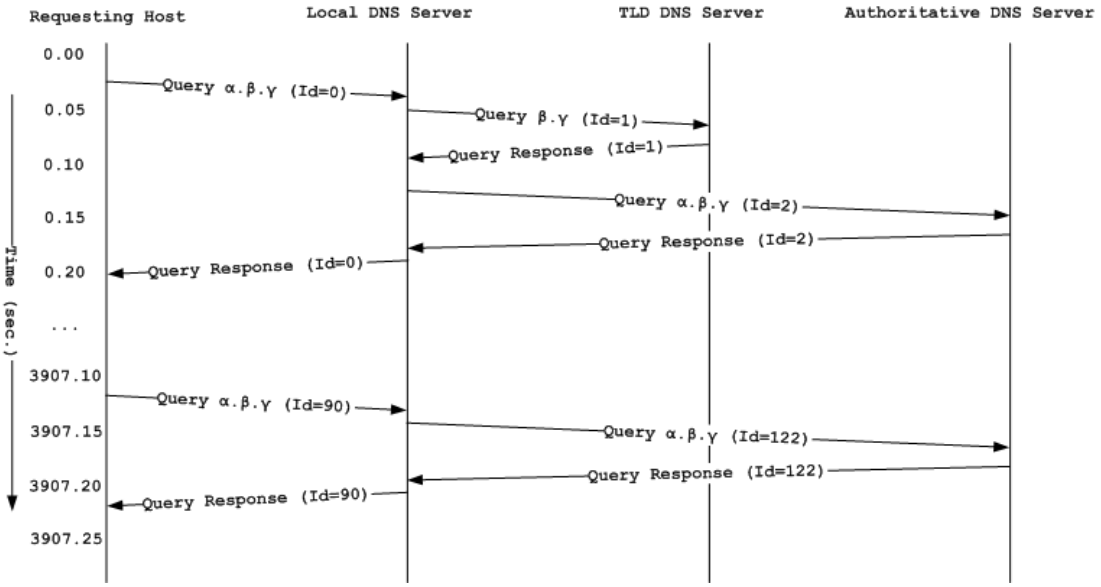


FIGURE 6.6: Example exchange of DNS messages (2)

6.4.4.2 DNS

Compared to HTTP, this scenario involves more than just two communicating entities. In order to completely resolve a DNS query, multiple entities might be contacted in the process. Still, comparable slices in DNS flows might be found.

The discussed requesting host might send a query for the same resource name at a later point in time, possibly because it lost the previously resolved IP address as a result of local cache characteristics. This could mean that a very similar set of messages would be exchanged between the referenced entities, resulting in similar slices. Fig. 6.6 shows such a scenario, where the requesting host wants to resolve the resource name $\alpha.\beta.\gamma$. Clearly, the flow consisting of the messages between the requesting host and the local DNS server can be separated into two slices that are very comparable. Both slices involve a request for the exact same resource, but have a different identifier. Since both slices take place between the same two entities and are the result of the same behavior of the requesting host, these two slices might be considered of the same slice type.

Note that different protocols require different approaches in order to properly compare slices.

6.4.5 Slice Set

Besides looking for slices within flows and comparing slices using slice types, slices in different flows might also be related. Multiple slices in possibly different flows that are to be considered related in their occurrence will form a slice set.

6.4.5.1 HTTP

Fig. 6.5 shows a trivial case, where only a single flow consists of four slices. Since the four slices do not appear to be related in their occurrence, due to the large time gap between them. The four slices should be considered as two separate slice set, each containing two slices that are related to the request of the single webpage and its related objects.

6.4.5.2 DNS

An example involving multiple flows is shown in Fig. 6.4. All of these messages shown are part of just a single DNS query. This means that, besides looking for slices within flows, one should also consider slices occurring in different flows. In this case, all three slices in the three different flows are part of the same DNS query operation and may be considered part of a single slice set. Subsequently, Fig. 6.6 shows two slice sets, where the second slice set contains all slices that contribute to the second DNS query issued by the requesting host.

6.4.6 Slice Set Type

As is the case with slices, also slice sets may be comparable. An exchange of messages between multiple entities that is the result of programmed behavior is possibly occurring more than once. Therefore, traces may contain comparable slice sets, since they may be the result of the exact same underlying behavior. The following will show how these slice sets might be comparable and therefore of the same slice set type in the case of the different protocols.

6.4.6.1 HTTP

Fig. 6.5 shows two slice sets, for which it is easy to see that these slice sets are of exactly one slice set type. A possible criterion for HTTP traffic in general might be that the various slices comprising the slice set all contribute to the download action for a particular webpage by the client, where some images, or other objects on the webpage might originate from potentially different servers.

6.4.6.2 DNS

The second set of messages (the second slice set) in Fig. 6.6 is highly comparable to the first slice set, since it involves resolving the exact same resource name. However, the sets need not have the exact same number of messages, because

some information might have been removed from the cache at one entity and not at another. This is shown in the second slice set, which contains an attempt to resolve $\alpha.\beta.\gamma$, where the local DNS server is still aware of the information provided by the TLD DNS server and therefore skips it. The second slice set is thus smaller in number of messages than the first one. Also, the identifiers used in the second set are different from the first set. This would leave the resource name as a possible means to compare these slice sets. In this case, the two slice sets might be considered of the same slice set type.

6.5 Relations in SNMP traffic

The preceding chapter has shown the possibility of identifying various types of relations in traffic in general, which suggests that finding elaborate relations in traffic is not limited to SNMP only. This chapter applies the now introduced relation types to SNMP traffic. The explanation given to each relation type in this context is a summary of a larger, more founded explanation given in BSc thesis [9]. This chapter is the foundation for the next step: being able to identify which SNMP traffic is to be considered periodic and which aperiodic.

6.5.1 Messages, Traces and Flows

The AIMS paper [11] gives a thorough description and exact definition of an SNMP message, trace and flow. Although the concept of each of these terms remains equivalent to the ones discussed in the general context, they are slightly more restricted in their properties in the case of SNMP.

For instance, an SNMP message is considered to carry a specific PDU with specific properties, like OIDs in the varbind list, but also the values related to that. Besides that, a request identifier and the type of message (e.g., get-request, or response) are also very important characteristics of an SNMP message.

An SNMP trace will consist of only SNMP messages. Like before, an SNMP trace has messages that are chronologically ordered (i.e., in the order in which they are recorded on a network segment).

An SNMP flow consists of SNMP messages that have two network layer endpoints in common. This limits an SNMP flow to a subset of SNMP messages in an SNMP trace that have been exchanged between two network layer endpoints, like an SNMP agent and a manager. An example is given in Fig. 6.7, which contains a visualization of an SNMP trace consisting of three flows.

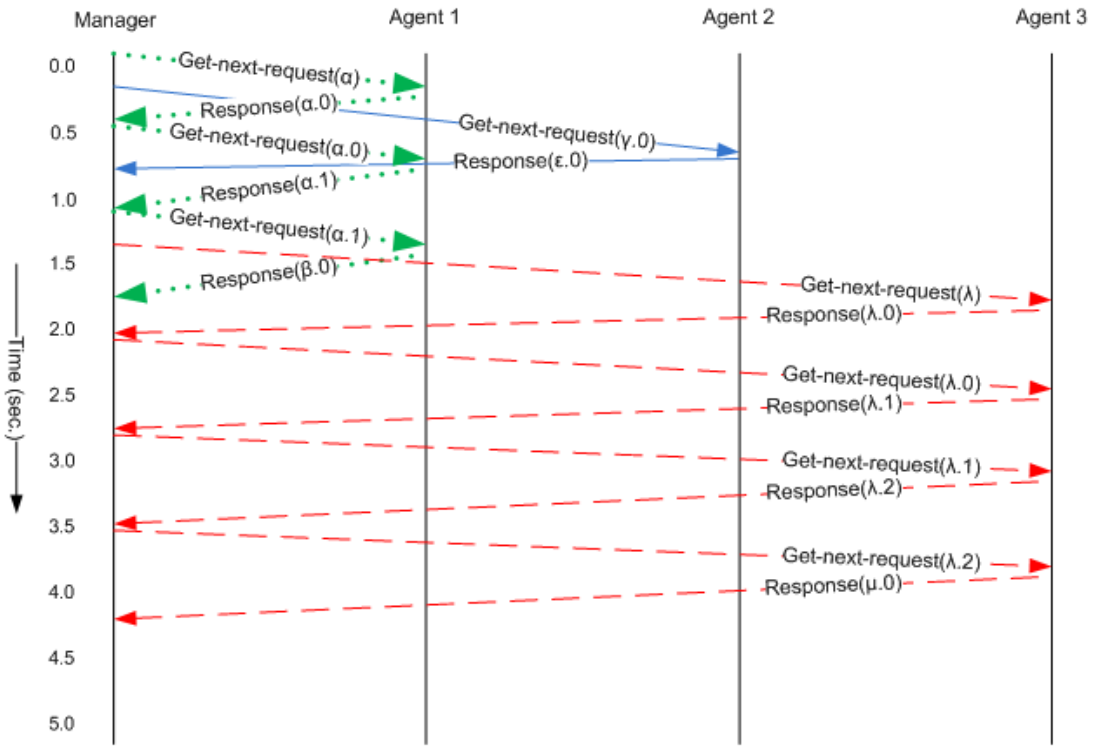


FIGURE 6.7: Trace consisting of three Flows

6.5.2 Slices

A flow is still too rough to say anything about periodicity, since a flow might contain messages that are the result of periodic and/or aperiodic behavior. We therefore need a relation type involving messages within flows. The SNMP slice is the answer to this question.

When considering a single manager and a single agent, like in Fig. 6.7 with the manager and agent 1, one might expect to see the manager to be programmed to poll that particular agent every few minutes. An SNMP slice involves the messages of a flow that are related to each other in a number of ways. Most notably, the requests must be of the same type (e.g., get-request, set-request) and the responses must contain a request identifier equal to a preceding request message sent by the manager.

The dotted lines in the Fig. 6.7 between the manager and agent 1 represent a number of messages of the flow between the two network layer endpoints. The first message is a get-next-request message sent to agent 1. It appears that the manager is interested in (some) of the values of the α table. The agent responds with $\alpha.0$, a message with the same request identifier as the first get-next-request. The manager carries on by sending a new get-next-request for $\alpha.0$, resulting in the value of $\alpha.1$. This goes on, until the manager receives a value for which the OID

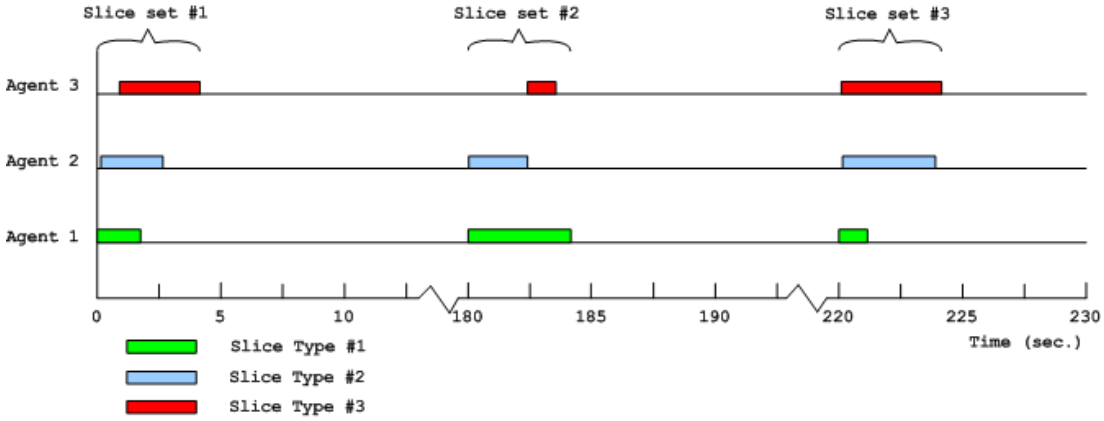


FIGURE 6.8: Slice Sets containing comparable Slices

(β) is of a different table than the preceding requests. Clearly, these messages are related and will form a slice. Similarly, there is also a single slice shown in the flows between the manager and agent 2 and agent 3 respectively.

6.5.3 Slice Types

SNMP slices have a number of important characteristics which allow for comparability among different slices. However, when should slices be considered equal (i.e., of the same slice type)?

The different slices must have the same 'meaning' (i.e., they should be the result of the same (programmed) behavior). Considering the fact that all of the slices shown in Fig. 6.7 are the result of the manager wanting to obtain specific information from the three agents, one might see similar slices in the respective flows multiple times. This, because the manager might be programmed to poll the agents every few minutes.

The comparable slices should therefore have similar OIDs, even though the number of messages might be different (tables may grow or shrink in size). Besides that, the comparable slices must be part of the same flow, because slices in different flows might be the result of different underlying behavior. Finally, comparable slices must have non-response messages that are of the same type. It makes no sense to compare slices with set-request messages with a slice containing get-next-requests, since they have a completely different goal. Also, retransmissions should be filtered out of slices before they are tested for equality, because they do not influence the meaning of slice. An extensive algorithm for determining comparability is described in [11].

6.5.4 Slice Sets

As Fig. 6.8 shows a fragment of a trace, where 9 slices can be identified and 3 slice types, it suggests that a single slice in a particular flow need not occur on its own. It may very well be that a manager is programmed to obtain specific information from a number of agents every few minutes. As a result, various comparable slices that are of the same polling instances, involving potentially different flows might occur multiple times in a trace. These apparently related slices form slice sets. In turn, the first slice set consists of the first 3 slices that appear to be occurring almost in parallel. The same can be said about the other 6 slices.

In order to find slice sets, it is necessary to look at slices that occur simultaneously or sequentially with not much time between any two slices. Another important criterion for a single slice to be considered part of a slice set, is the initiator of the slices. Slices initiated by two different managers do not belong to the same underlying behavior, like a polling instance. Therefore, slices belonging to the same slice set are all initiated by the same network layer endpoint.

It should be noted that finding slice sets in reality is more complex. The chosen approach encompasses a broader look than just slices and also incorporates slice set types in order to find slice sets.

6.5.5 Slice Set Types

Slice sets may also be comparable, because they may be constructed from slices that are of the same slice type. Fig. 6.8 shows three slice sets that all contain slices that are of the same slice type. As a result, these slice sets are comparable and are therefore of the same slice set type.

In reality it may be difficult to find slice sets that contain the exact same number of slices and that are of the same slice type. This can be because of numerous reasons, like an agent that is offline for a while. During that time, the manager may attempt to request information from that agent every few minutes, but because the manager does not get a response to its very first request, it may not go on querying other variables of that agent. This might, to some extent, result in very different slice sets that are still the result of the same behavior of the manager. As a result, some flexibility is to be introduced in the process of identifying comparable slice set per slice set type.

Our approach attempts to find slices of a particular slice type that appear to occur within a specific time frame of each other throughout the trace. An example is shown in Fig. 6.9, which shows the slices in a trace that are all initiated by the same network layer endpoint and where some slices occur parallel to each other.

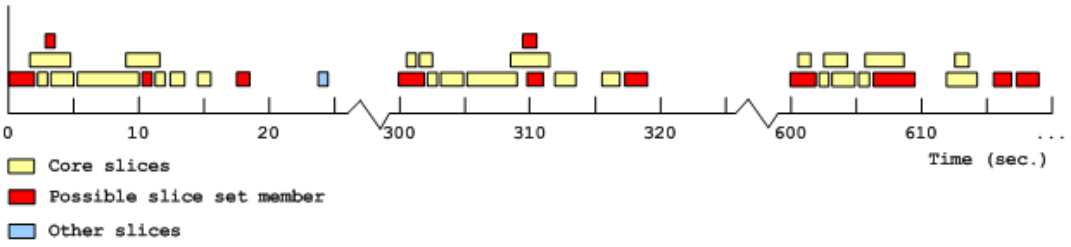


FIGURE 6.9: Finding related slices

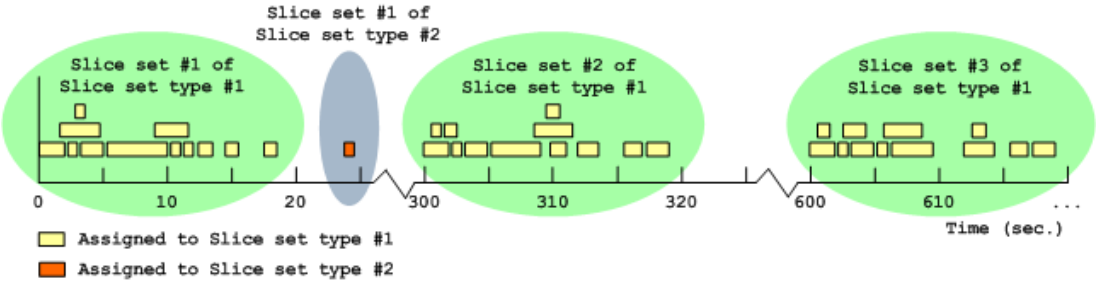


FIGURE 6.10: Slice sets of a specific slice set type

The first step therefore involves finding the so called 'core' slices, which always occur within a specific time frame of each other, are of the same slice type respectively and should therefore be considered the basis of the slice sets that will be of the same slice set type. After that, one should look for slices of slice types that occur in almost all cases within the time frame of the already marked core slices. This is to allow for scenarios where, for instance, an agent might be offline for a while.

After this, a number of comparable slice sets result, as is shown in Fig. 6.10. In this case only a single slice in the trace is not considered part of any of these slice sets. That particular slice may therefore form a different slice set of a different slice set type.

6.5.6 Discussion

It should be noted that the presented description of finding related messages is in reality more complex. For instance, the actual definition of a slice, as stated in [11], is quite elaborate and constructed based on possible scenarios that might occur in real world SNMP traffic. For instance, the time between the request and the response with the same request identifier is to be limited in order for them to be considered related. Also, a distinction in the comparison process is made between various messages types: a slice containing a trap message will have different characteristics than a get-bulk slice.

Similar cases can be made for other relations, like the slice set and slice set type. Thesis [9] describes, amongst many other considerations, situations where multiple managers might operate from a single IP address, or have polling configurations resulting in irregularly occurring slice types within slice sets of a particular slice set type. The interested reader is referred to [9] and [11] for a more elaborate description of all of the considerations.

6.6 Determination of Periodicity

The ability to find comprehensive relations, like the ones described so far, allow for analysis of the SNMP traffic at a relatively high level: the slice sets per slice set type. This abstract view eases the analysis process significantly, compared to a per-message analysis approach. This advantage is used in our approach of identifying periodic and aperiodic SNMP traffic.

6.6.1 Identifying periodicity

Our approach attempts to find periodicity by considering all slice sets per slice set type. All of these slice sets are considered comparable and as a result may to some extent show periodic behavior. Consider again the example where a manager polls a number of agents every few minutes, resulting in potentially a single slice set per polling instance. If this manager is programmed to perform that specific polling instance every x minutes, then one should observe that in the properties of the respective slice sets of a particular slice set type.

It may be incorrect to assume that the above mentioned slice sets would have clear intervals by merely considering the starting time (the time of the chronologically first message of a slice in a slice set), that is exactly `x` minutes later than a preceding slice set. One of the reasons for deviations in the expected start time might be numerous. One might be that it is unknown when the manager restarts its polling timer. It may restart its timer at the beginning of a polling instance, when it has received the last response message from the last polled agent, or possibly some other point in time during the polling instance.

Another reason for deviations is that slices of a particular slice set type might be the result of multiple polling configurations (i.e., the manager is set to poll every 4 minutes, but also every 30 minutes). Similar cases might go hand in hand with erroneous configurations or a reset of managers.

Also, polling instances may be induced manually, resulting in a possible mixture of periodic and aperiodic slice sets that are all of the same slice set type.

And finally, network delays may differ over time resulting in moderate starting time variations from the perspective of the recording unit of the trace. Moreover, traces may not be complete. For instance, a manager might be turned off for a few minutes, resulting in a gap in the trace and potentially a different starting moment for the polling timer when it restarts.

6.6.2 Example

Slice Set (#)	**Start Time (s)**
1	0.2
2	150.3
3	300.2
4	450.7
5	505.1
6	600.5
7	750.9
8	900.6

TABLE 6.1: Slice sets of a particular Slice Set Type

Table 6.1 shows 8 slice sets that are all of the same slice set type. Our approach would attempt to find periodic behavior first by considering the various possible polling timer reset points. For simplicity, only the start time of a slice set is shown and considered here.

The time between the start time of the first slice set and the second is 150.1 seconds. Based on this first interval, an attempt is made to find a third slice set that started 150.1 later than the second. Because of the reasons mentioned before, a limited margin is applied to this 150.1 seconds in the search process. As a result, the third slice set would be found. Then the interval between the first and second and second and third slice sets is averaged, resulting in an average interval of 150.0 seconds. This average is subsequently used in the search process for a potential fourth slice set that is to be contributing to this particular interval. That search would look for a slice set starting at around 450.2, which would find slice set 4. This process continues until all but the fifth slice set is considered to be contributing to periodic behavior with an interval of about 150 seconds. How the margins and other details of the seek algorithm are to be chosen is discussed in [9].

The process would normally go on with the remaining slice sets and attempt to find a common interval between them as well. In the end, for each slice set type, only the slice sets that are to be considered aperiodic remain. Note that a significant number of slice sets per slice set type contributing to a particular

interval is necessary. This should not be a problem, since traces are usually quite lengthy in duration, allowing for a large enough sample space.

In this case the remaining fifth slice set remains and is therefore considered aperiodic, since it cannot be linked to any other slice set of this particular slice set type with a specific interval. This slice set may for example be the result of a manager being ordered to poll its agents after receiving a manual command from the operator, not the polling timer.

6.6.3 Discussion

Finding periodicity in slice sets of a particular slice set type as described in this chapter is just one of potentially many ways of detecting intervals between slice sets of a slice set type this way. In contrast to the observations made about Fourier analysis in chapter 2, Fourier analysis might assist in finding intervals between slice sets, though this has not been explored.

6.7 Evaluation

This chapter describes the testing of the implemented approach by applying it to both artificial and lab traces.

6.7.1 Toolset

The described process has been implemented in the form of a set of Perl-scripts. These scripts use as input a csv (comma separated values) trace file that is the output of an already existing tool: snmpdump. Trace files are usually recorded in the pcap format, which is not very practical for analysis. Therefore, snmpdump (a tool developed as part of [3]) was chosen to convert these pcap files to csv files, which can subsequently be analyzed by our scripts. The set of scripts start by identifying the various relation types in the trace, followed by finding periodicity between slice sets per slice set type.

6.7.2 Artificial trace test

In order to test our implementation, we used an artificial trace file. This csv trace file was constructed manually and consisted of all kinds of exceptional cases one might find in real world traces. This included all possible message types that are

defined in the SNMP standards [14], [15] like get-, get-next-, get-bulk-, set-, inform-requests, but also trap messages of various SNMP versions. Subsequently, the respective response messages contained OIDs that varied from normal sequences to random sequences and simulations of gaps in tables. Numerous other cases are described extensively in the 'considerations' chapters found in [11].

The trace consists of 1 flow, which in turn consists of 12 slices. These 12 slices are of different slice types and should result in 1 slice set of 1 particular slice set type. Applying this trace file as input to our implementation yields the proper number of flows, slices, slice types, slice sets and slice set types.

6.7.3 Lab trace test

Besides this artificial trace, a lab trace is used to verify the approach. The lab setup consists of a single SNMP enabled router and a computer running SNMP manager Cacti on Debian. Cacti was configured to query a number of IF-MIB on that router every 5 minutes. The trace has a duration of about 2.5 days.

The recorded trace is first converted from a pcap format to a csv format using snmpdump. The csv file is subsequently analyzed, resulting in a separation of the trace into the expected relations.

Since the trace is made on a subnet involving a single manager and a single agent, only one flow is identified. That flow consists of 13502 slices that can be mapped to just 20 slice types. This means that a very large number of slice types are to be considered comparable, which is to be expected in this case. The determination of slice set types yields just a single one, with 677 slice sets of that particular type. It can be seen that all slices making up a single polling instance are part of a single slice set, because 677 * 5 minutes is approximately the duration of the trace. Finally, attempting to find periodicity gives the observation that all slice sets of this single slice set type contribute to a single interval: 5 minutes.

These two traces indicate that our approach seems to be correct.

6.8 Periodicity in real world traces

In the past few years a number of research institutes have started to collect real world SNMP traces from different networks from around the world at varying network sizes. These traces have been analyzed for many different characteristics, as is for instance described in [3]. Some of these SNMP traces have also been analyzed for periodicity using our toolset.

Property	**Trace 1**	**Trace 2**	**Trace 3**	**Trace 4**
Messages processed	71501	15099	361000	1121000
Unhandled messages	544	0	0	0
Slices	8084 total 5321 get-next 2636 get 127 set	7504 total 192 get-next 4527 get 2785 get-bulk	20940 total 5157 get-next 1783 get	573405 total 6 get-next 573395 get 4 trap
Slice Types	5503 total 3726 get-next 1650 get 127 set	157 total 4 get-next 95 get 58 get-bulk	1277 total 57 get-next 1220 get	136 total 4 get-next 129 get 3 trap
Slice sets	39	737	4734	4239
Slice set types	29	16	43	8
Periodic slice sets	0	147	72	4229
Aperiodic slice sets	39	0	4662	0
Ambigious slice sets	0	590	0	10

TABLE 6.2: Periodicity in traces

Table 6.2 shows a summary of the analysis results of four of these traces. It should be noted that traces 3 and 4 have only been partly analyzed, because the analysis process takes a very large amount of time. For instance, in the case of trace 3, only the first 361000 SNMP messages of the trace have been analyzed, which took about 7 processing days on a dual Xeon processor computer with 4 GB of RAM. In contrast, trace 1 required almost 2 days to be processed completely.

Following is a more in-depth description of the analysis result for each of these four traces.

6.8.1 Trace 1

This trace is recorded on a national research network, which contains 24 hours of recorded SNMP traffic. All SNMP messages in this trace were processed, since it is a relatively small trace file.

There were 544 SNMP messages that could not be assigned to any slice. These are all response messages. This could be the result of the fact that some requests have not been recorded due to network characteristics and are therefore not included in the trace file. Therefore, the respective response message cannot be assigned to slices, because there is no existing slice with the respective request listed.

Also, another important aspect of the analysis result is the relatively large number of slice types found with respect to the number of found slices. This suggests that there is a large number of incomparable slices in this trace file.

Finally, the table shows that no slice sets have been marked as periodic, so no intervals have been found. All slice sets are found to be aperiodic, because only one slice set type describes three slice sets, which do not have an identifiable interval. All other slice set types describe only two or just one slice set, for which automatically aperiodicity is concluded. A different report [10] on this particular trace confirms this finding. Also, an explanation from the network manager suggests that configuration took place at the time of the trace recording, resulting in manually generated (i.e., aperiodic) polling instances.

6.8.2 Trace 2

The second trace is recorded at a server-hosting provider, which contains 4 hours of recorded SNMP traffic. In contrast to the first trace, this trace has a small number of slice types found with respect to the number of slices. This suggests that there is a large number of slices in this trace file that are of the same slice type.

The table also shows that 16 slice set types were found, while in reality there may have been fewer. This, because manual inspection of this trace file suggests that there are very large gaps between some slices initiated from one particular IP address, which may in reality still be part of the same polling instance. In some cases, this gap is larger than 60 seconds. Besides load spreading by an intelligent manager application, this phenomenon could also be the result of the possibility that some traffic, involving certain polling instances, has never been seen by the trace file recording unit, due to network characteristics.

Another important observation regarding this trace file, is that information about the involved network elements suggests that the primary manager of that network operated on a system with three different IP addresses. The number of slices per initiator that were detected in this trace confirms this. It should be added that even after considering the three initiators to be the same, there would still be very large gaps between some slices that may in reality belong to a single polling instance.

Finally, our script execution shows that one slice set type, involving 49 slice sets, initiated by one particular IP address, are all considered to be periodic. An interval of 300 seconds was detected and no slice sets of this slice set type were considered aperiodic. The remaining slice set types were initiated by either of the other two initiators (i.e., IP addresses). The respective slice sets involve those that should actually be considered part of just one slice set type, as a result of the described anomaly involving the single manager with three IP addresses. But, because the cause of this phenomenon remains unknown, some slice sets of these slice set types have been marked as ambiguous, instead of either periodic or aperiodic. In total 92% of all slices are assigned to slices sets that are marked periodic and 8% are

marked ambiguous. If the cause of this phenomenon can be determined, then it may be that all slices can be considered part of slice sets that are marked periodic.

6.8.3 Trace 3

The third trace was recorded on a university network, which contains 6.5 days of recorded SNMP traffic. Only the SNMP messages that occurred at the first recording hour of this trace have been analyzed, due to large processing time for this trace.

Besides the trivial observations regarding the different relations between slices and slice types, this trace shows 4 different intervals that can be found in the periodic slice sets. 12 Slice sets show an interval of 900 seconds, 20 show 180 seconds, 34 show 300 seconds and 6 show an interval of about 55 seconds.

Also, although by far the greatest portion of all slices is assigned to slice sets that are periodic, there is still a large number of slice sets that are considered to be aperiodic. In the case of this trace, there is one particular reason for this. Many slices that are of a particular slice type occur in some polling instances more than once. Some incorrectly set parameters used in the analysis process may have therefore affected the results, which incorrectly suggests a high number of aperiodic slice sets. This, because no interval can be determined for these irregularly occurring slice sets. It is expected that different parameter values would show a more realistic result. This shows the significant influence analysis parameters can have in this process.

6.8.4 Trace 4

The fourth trace was recorded on a German governmental network, which contains 24 days of recorded SNMP traffic. Only the SNMP messages occurring in the first day of the trace have been analyzed.

The results show a small number of slice types found with respect to the number of slices. This suggests that there is a very large number of slices in this trace file that are of the same slice type.

Another analysis result is that the largest slice set type, involving 4229 slice sets, are all periodic. All of these slice sets contribute to a single interval of 20 seconds. The remaining 10 slice sets were initiated by different network element, forming slice set types which each encompass 3 slice sets or less. No periodicity could be found in any of these slice sets, resulting in 10 aperiodic slice sets.

6.8.5 Discussion

The analysis of the first two traces show the most interesting of all of our analysis results. Our results in general indicate that our approach seems to be a proper one, except that there are still some irregularities in real world traces that our approach does not account for. This is clearly indicated by the results of the second trace and also to some extent of trace 4. Manual inspection of these traces suggest that the slice sets marked as ambiguous seem to be part of polling instances with a periodic character. The fact that these are actually marked as ambiguous is the result of the fact that the relevant slice sets do not appear to occur regularly enough within these polling instances to be considered part of them.

6.9 Conclusions and Future Work

SNMP traces contain messages that contribute to either periodic or aperiodic behavior. Being able to separate these two kinds of traffic allows for specific research on either of the two. This could, for instance, assist network operators in analyzing network characteristics. However, until now no known approach is available that can perform a complete separation on a message level for SNMP.

Which approaches to separate periodic from aperiodic SNMP traffic are possible?
As explained in section II, three approaches can be considered to separate periodic from aperiodic SNMP traffic. Approaches include using Fourier analysis, or reverse engineering of known SNMP agent and manager implementation. The first one does not work, because it is not possible to determine exactly which SNMP messages in a trace are responsible for specific intervals. The second approach has as disadvantage the necessity to reverse engineer a large number of applications in order to obtain the characteristics of them. Even if those characteristics are known, it would still be cumbersome to distill the patterns from actual traces and determine periodicity. What remains is a possible third approach, which could be seen as a generic approach involving finding various kinds of relations between SNMP (sets of) messages in traces, based on protocol characteristics. Determining actual (a)periodicity is done by detecting intervals between these relations.

Which SNMP properties characterize a working solution?
Our approach first attempts to find a large number of related SNMP messages, including slices, in accordance with the definitions stated in [11]. We go on with looking for even larger sets of messages, such as slice sets that are of a specific slice set type, which will form a basis for determining periodicity.

How can periodicity be determined?
Periodicity is determined by recognizing specific intervals between the slice sets that are of the same slice set type. This process is described is section 5 and involves taking the time between the occurrence of the chronologically first slice set and the chronologically second slice set that are of the same slice set type. Based on the time between these two, a third slice set is looked for. An interval is discovered if and only if a substantial number of slice sets can be found to be contributing to a single interval this way. If it fails, the process will be repeated and starts considering the first and third slice set. This continues until all possible intervals have been identified between slice sets that are of the same slice set type. Although not described in this paper, it may however also be possible to apply Fourier analysis on slice sets of a slice set type in order to identify these intervals.

What periodicity can be found in lab and real world traces?
The implementation of the algorithm into a toolset of scripts and the application of the scripts on SNMP traces, as described in section 6, show that the artificial and lab traces were successfully analyzed. Moreover, the real world trace analyzes show that almost all periodic SNMP traffic could be separated completely and without any uncertainty from the aperiodic SNMP traffic. However, a small percentage of the SNMP traffic is considered ambiguous and therefore no complete separation of the two types of SNMP traffic could yet be attained for every available SNMP trace.

Even though the determination of periodicity within SNMP traffic is possible, it still deserves more research. First of all, more research is needed on the specifics of SNMP traffic that is marked 'ambiguous'. Results of such research could be used to improve the algorithm and the related toolset in order to achieve separation of periodic and aperiodic SNMP traffic in all cases. At the moment we would consider incorporating knowledge of the current SNMP implementations (using limited reverse engineering) as a possible solution. Still, the mentioned 'ambiguous' traffic should be analyzed extensively, even after obtaining knowledge of the current engine implementations, because it may still be difficult to state with a high level of certainty that it is actually contributing to either periodic or aperiodic traffic. Besides that, it would also be of much interest to understand more about the exact causes and characteristics of aperiodic SNMP traffic.

Acknowledgment

The work reported in this paper was supported by the EC IST-EMANICS Network of Excellence (#26854). We would like to thank the various network operators who gave us the possibility to collect and analyze traces.

6.10 References

1. Case, J., Mundy, R., Partain, D., Stewart, B.: Introduction and Applicability Statements for Internet Standard Management Framework. RFC 3410 SNMP Research, Network Associates Laboratories, Ericsson (December 2002)
2. Harrington, D., Presuhn, R., Wijnen, B.: An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. RFC 3411, Enterasys Networks, BMC Software, Lucent Technologies (December 2002)
3. Schönwälder, J., Pras, A., Harvan, M., Schippers, J., van de Meent, R.: SNMP Traffic Analysis: Approaches, Tools, and First Results. In proc. 10th IFIP/IEEE International Symposium on Integrated Network Management (May 2007)
4. Chen, Y., Hwang, K. and Kwok, Y.: Filtering of Shrew DDoS Attacks in Frequency Domain. In The IEEE Conference on Local Computer Networks 30th Anniversary (LCN05) (November 2005). pp. 786-793
5. Cheng C., Kung, H.T. and Tan, K.: Use of Spectral Analysis in Defense Against DoS Attacks. In Global Telecommunications Conference, 2002. GLOBECOM 02. IEEE (November 2002)
6. Jarzabek, S., Woon, I.: Towards a precise description of reverse engineering methods and tools. In Software Maintenance and Reengineering, 1997. EUROMICRO 97., First Euromicro Conference on (March 1997)
7. Lakhina, A., Crovella, M., Diot, C.: Diagnosing Network-Wide Traffic Anomalies. SIGCOMM, 2004. (September 2004)
8. Harvan, M.: SNMP Traffic Measurement and Analysis Seminar Report for Networks and Distributed Systems Seminar. (September 2006)
9. van den Broek, J.G.: Periodicity of SNMP Traffic. BSc Thesis (August 2007)
10. Grondman, I.: Identifying short-term periodicities in Internet traffic. BSc Thesis (December 2006)
11. van den Broek, G., Schönwälder, J., Pras, A., Harvan, M.: SNMP Trace Analysis Definitions. In proc. 2nd International Conference on Autonomous Infrastructure, Management and Security (July 2008)
12. RFC 2616: Hypertext Transfer Protocol HTTP/1.1 (June 1999)
13. RFC 1035: Domain Implementation and Specification (November 1987)
14. RFC 1157: Simple Network Management Protocol (SNMP) (May 1990)
15. RFC 2574: User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3) (April 1999)